PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

PLATAFORMA PARA INJEÇÃO DE FALHAS EM SYSTEM-ON-CHIP (SOC)

MARCELO MALLMANN DIAS

PORTO ALEGRE
2009

MARCELO MALLMANN DIAS

PLATAFORMA PARA INJEÇÃO DE FALHAS EM SYSTEM-ON-CHIP (SOC)

Dissertação apresentada como requisito para obtenção do grau de Mestre, pelo Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia, Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fabian Luis Vargas.

Porto Alegre

2009

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.

- *Douglas Adams*

Agradecimentos

Agradeço aos meus pais por despertar em mim o interesse pelo mundo e pelo apoio incondicional;

À Lila, pelo apoio, carinho e principalmente pela paciência;

Aos meus orientadores, Fabian e Letícia, pelo interesse, esforço e oportunidades;

Aos amigos do SiSC, pelo carinho, amizade, pela saudade de um chimas no meio do laboratório;

Aos engenheiros Chris Dennis e Steve Rhoads, monstros sagrados do VHDL, pela iluminação.

Ao CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico e a CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pelo financiamento deste trabalho.

Resumo

O aumento do número de sistemas computacionais embarcados sendo utilizados em diversos segmentos de nossa sociedade, de simples bens de consumo até aplicações críticas, intensificou o desenvolvimento de novas metodologias de teste e técnicas de tolerância a falhas capazes de garantir o grau de confiabilidade esperado os mesmos. A injeção de falhas representa uma solução extremamente eficaz de avaliar metodologias de teste e técnicas de tolerância a falhas presentes em circuitos integrados complexos, tais como *Systems-on-Chip* (SoCs). Este trabalho propõe uma nova plataforma de injeção de falhas que combina conceitos relacionados a técnicas de injeção de falhas baseadas em hardware e em simulação. Esta nova plataforma proposta é capaz de injetar diferentes tipos de falhas nos barramentos presentes em diversos componentes funcionais de um SoC descrito em VHDL. O uso de sabotadores controlados por um gerenciador de injeção de falhas instanciado no mesmo FPGA que o sistema a ser avaliado é capaz de prover uma alta controlabilidade aliada a baixa intrusividade e uma grande gama de modelos de falhas. Além disso, é importante salientar que a plataforma proposta representa uma solução fácil no que diz respeito à configuração e automação de experimentos de injeção de falhas.

Abstract

The increasing number of embedded computer systems being used in several segments of our society, from simple consumer products to safety critical applications, has intensified the study and development of new test methodologies and fault tolerance techniques capable of assuring the high reliability expected from those systems. Fault injection represents an extremely efficient way of the test and the fault-tolerant techniques often adopted in complex integrated circuits, such as Systems-on-Chip (SoCs). This work proposes a new fault injection platform that combines concepts related to hardware-based and simulation-based fault injection techniques. This new platform is able to inject different kinds of faults into the busses present in several functional components in a VHDL described SoC. The use of saboteurs controlled by a fault injection manager instantiated in the same FPGA as the target system provides high controllability coupled with low intrusiveness and a wide range of possible fault models. Moreover, it is worth noting that the proposed platform represents an easy solution with respect to the configuration and automation of fault injection campaigns.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

CLB	*Configurable Logic Block*
FPGA	*Field Programmable Gate Array*
DOCTOR	*Integrated Software* Fault Injection *Environment*
FIST	*Fault Injection for Study of Transient fault effect*
FITO	*FPGA-based fault Injection Tool*
I/O	*Input – Output*
LUT	*Look-Up Table*
MEFISTO	*Multi-level Error/*Fault Injection *Simulation Tool*
MIPS	*Microprocessor without Interlocked Pipeline Stages*
NoC	*Network-on-Chip*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
SiSC	*Sinais Sistemas e Computação*
SoC	*System-on-Chip*
UART	*Universal Asynchronous Receiver Transmitter*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very-high-speed integrated circuits*

1. Introdução

Atualmente é possível observar que a presença de sistemas computacionais embarcados representa uma tendência extremamente forte e crescente nos mais variados segmentos de nossa sociedade, ou seja, é possível encontrar sistemas embarcados em aplicações que vão desde um simples telefone celular até um complexo sistema de tráfego aéreo. Nesse contexto, observa-se que seu escopo, complexidade e disseminação aumentaram dramaticamente nos últimos anos, visto que vários e diversos segmentos tais como industriais, comerciais, médicos e de transporte, dependem diretamente de tais sistemas. Assim, de acordo com os requisitos estipulados e as funcionalidades a serem implementadas, o projeto de sistemas embarcados torna-se cada vez mais complexo e seu desempenho e confiabilidade cada vez mais críticos e importantes. Além disso, o custo e as conseqüências relacionadas a uma possível falha podem assumir dimensões que vão desde meros incômodos até enormes catástrofes.

Entretanto, a prática nos mostra que dificilmente podemos garantir que tais sistemas sejam livres de falhas. Basicamente, as falhas em sistemas embarcados podem ser o resultado de um erro cometido durante o projeto, do desgaste físico intrínseco ao sistema ou, ainda, podem ser causadas por diferentes fatores externos durante o período de funcionamento dos mesmos. Neste contexto, as técnicas de injeção de falhas representam uma solução extremamente viável durante a fase de desenvolvimento do projeto e indispensável quando queremos garantir que sistemas embarcados continuem funcionando mesmo quando inseridos em ambientes considerados hostis. Entretanto, convém salientar que técnicas de tolerância a falhas precisam ser avaliadas no que diz respeito a capacidade de detecção de dfeitos e *overheads* inseridos com o objetivo de determinar o quanto as mesmas podem agregar de robustez aos sistemas. Tal avaliação pode ser feita através da utilização de metodologias de injeção de falhas que consistem em um mecanismo capaz de injetar diferentes tipos de falhas nos mais diversos blocos que compõem os sistemas embarcados e de proverem a observabilidade e controlabilidade necessárias para o experimento. Nesse contexto, o uso de técnicas de injeção de falhas emergiu

como uma ferramenta extremamente importante para a avaliação de técnicas de tolerância a falhas utilizadas em sistemas embarcados que implementam diferentes aplicações críticas com o objetivo de aumentarem a confiabilidade das mesmas.

Diversas técnicas de injeção de falhas já foram propostas e utilizadas na prática. Estas técnicas podem ser basicamente, divididas em três grupos: (1) técnicas baseadas em simulação são capazes de observar e injetar qualquer tipo de falhas em qualquer ponto do sistema, mas sua complexidade cresce com o nível de detalhamento do sistema simulado. (2) técnicas baseadas em *hardware* são capazes de reproduzir mais fielmente o mundo real e, conseqüentemente produzem uma avaliação mais confiável do comportamento do sistema alvo, mas implica em um custo mais elevado. (3) técnicas baseadas em *software* são flexíveis e baratas, mas envolvem alterações no comportamento do sistema alvo.

Em vista do exposto, o principal objetivo deste trabalho de mestrado é propor uma nova plataforma de injeção de falhas em S*ystems-on-Chip* (SoCs) que explora o uso de um núcleo injetor de falhas desenvolvido em lógica programável integrado ao sistema alvo juntamente com um *software* de controle. A plataforma proposta tem por base privilegiar a injeção de falhas de forma transparente, sem que seja necessário interromper o funcionamento normal do sistema para que seja efetuada a injeção da falha. Por conseqüência, a plataforma de injeção de falhas desenvolvida pode ser utilizada para avaliar a robustez de sistemas em tempo real, sem comprometer as restrições temporais impostas pelos mesmos.

A plataforma de injeção de falhas proposta está disponível para uso do grupo de pesquisa SiSC/PUCRS (Sinais, Sistemas e Computação, da Pontifícia Universidade Católica do Rio Grande do Sul) como um complemento aos métodos de injeção de falhas utilizados atualmente. Especificamente, a plataforma proposta representa uma alternativa aos métodos de injeção de falhas baseados em hardware disponíveis dentro do grupo SiSC capaz de prover a controlabilidade e observabilidade exigidas para avaliar uma série de metodologias de teste e técnicas de tolerância a falhas.

1.1. Objetivos

O principal objetivo deste trabalho é especificar, implementar e validar uma plataforma de injeção de falhas capaz de injetar diferentes tipos de falhas nos barramentos associados aos mas variados módulos funcionais presentes em SoCs. Especificamente serão abordados os seguintes tópicos:

- Desenvolvimento de uma nova plataforma de injeção de falhas que possa ser utilizada durante a fase de projeto de sistemas embarcados de tempo real;
- Verificação da possibilidade de injeção de falhas nos barramentos de um SoC sem que seja necessário interromper o seu funcionamento para executar a injeção da falha;
- Será investigada a possibilidade de injeção de falhas nas memórias presentes em SoCs a partir da utilização dos barramentos que conectam as mesmas com o microprocessador;
- Criação de uma arquitetura que permita a fácil especificação e implementação de diferentes modelos de falhas pelo usuário;
- Automação do processo de criação e gerenciamento de uma campanha de injeção de falhas.

1.2. Organização

Este trabalho foi organizado da seguinte forma:

- Capítulo 2, Modelos de Falhas: Apresenta uma série de conceitos e definições preliminares relacionadas às metodologias de injeção de falhas. Além disto este capítulo descreve os principais defeitos físicos relacionados a módulos de memória e barramentos, assim como apresenta os principais modelos de falhas adotados relacionando-os com os modelos de falhas adequados;
- Capítulo 3, Técnicas de Injeção de Falhas: Este capítulo apresenta o estado-da-arte em técnicas de injeção de falhas, classificando-as e apresentando as principais referências existentes na literatura;

- Capítulo 4, Plataforma de Injeção de Falhas Proposta: Apresenta o ambiente de injeção de falhas proposto e descreve a plataforma de testes, o bloco injetor de falhas e sistemas auxiliares desenvolvidos;
- Capítulo 5, Validação e Resultados Experimentais: Apresenta o procedimento utilizado para validar a metodologia proposta e uma análise comparativa da mesma em relação às técnicas de injeção de falhas presentes na literatura;
- Capítulo 6, Conclusão: apresenta as principais conclusões e relaciona os trabalhos que podem ser desenvolvidos como continuação dessa proposta.

2. Modelos de Falhas

A avaliação de metodologias de teste e de técnicas de tolerância a falhas é feita com base na utilização de modelos de falhas definidos a partir de defeitos físicos reais observados em circuitos integrados. Segundo Bardell [1], um modelo de falha especifica a série de defeitos físicos que podem ser detectados através de um procedimento de teste. Um bom modelo de falha, segundo Stroud [2], deve ser computacionalmente eficiente em relação ao dispositivo de simulação e refletir fielmente o comportamento dos defeitos que podem ocorrer durante o processo de projeto e manufatura, bem como o comportamento das falhas que podem ocorrer durante a vida útil do sistema. Estes modelos são utilizados na emulação de falhas e defeitos durante as etapas de simulação e validação do projeto. Bem como durante a realização de experimentos de injeção de falhas realizados com o objetivo de avaliar a robustez de metodologias de teste e técnicas de tolerância a falhas.

Neste capítulo serão apresentados os principais conceitos abordados nesta dissertação de mestrado e os diversos modelos de falhas definidos para barramentos e módulos de memórias presentes em um SoC.

2.1. Falha-Erro-Defeito

Segundo Laprie [3] uma falha (*fault*) é identificada ou hipotetizada como a causa de um erro. Uma falha pode ser vista simplesmente como a conseqüência de um defeito em algum outro sistema (incluído o projetista) que provê, ou está provendo um serviço para o sistema. Uma falha ativa é uma falha que produz um erro. Causas comuns de falhas incluem desgaste, envelhecimento e interferência externa, humana ou ambiental, nos componentes de um sistema. Por isso, diz-se que falhas estão associadas ao universo físico.

Falhas podem ser classificadas, no domínio do tempo, em permanentes, transientes ou intermitentes. Falhas permanentes são geradas durante o processo de fabricação ou durante a vida útil do sistema (e.g: curto-circuitos ou conexões abertas). Falhas transientes ocorrem durante o funcionamento do sistema, podem ter curta ou longa duração e são geradas por fenômenos aleatórios, como interferência eletromagnética. Finalmente, falhas intermitentes são definidas pela sua ocorrência temporária e repetida causada por algum fenômeno externo, como variações na temperatura ou vibrações mecânicas.

Erro (*error*), ou estado errôneo é um estado do sistema decorrente de uma falha capaz de gerar um defeito. Erros estão associados ao universo da informação. Alguns tipos de fenômenos tais como rajadas de radiações eletromagnéticas, podem causar erros, simultaneamente, em mais de um componente do sistema. Esses tipos de erros são chamados de múltiplos erros associados. Por sua vez, erros que afetam somente um componente do sistema são chamados de erros simples.

Finalmente, defeito (*failure*) é um desvio no comportamento do sistema em relação ao especificado conforme percebido por seu usuário. Um serviço é definido como defeituoso quando ele não completou a especificação funcional como deveria, ou quando a especificação não foi adequadamente descrita durante a fase de especificação do sistema. O desvio do serviço correto pode assumir diferentes formas, que são chamadas de modos de defeito de serviço (*service failure modes*) e que são classificados de acordo com a gravidade do defeito. Considerando que um serviço é uma seqüência de estados externos do sistema, um defeito no serviço significa que pelo menos um ou mais estados externos do sistema se desviaram do estado correto do serviço.

A dinâmica da propagação de uma falha em um defeito é mostrada na Figura 2.1:

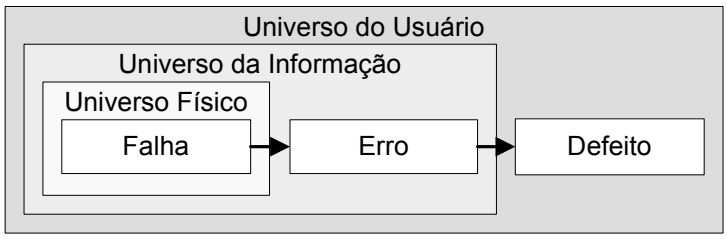

Figura 2.1 – Modelo falha-erro-defeito [3].

Assim, a causa de um erro é uma falha. O tempo entre a ocorrência da falha e a percepção de um estado errôneo é chamado de latência de falha. Erro é a manifestação de uma falha, mas não necessariamente leva a um defeito. Múltiplos erros podem ser originados a partir de uma mesma falha. Se estiverem presentes todos os mecanismos necessários à propagação do erro, este será detectado após um tempo chamado de latência de erro. Erros não tratados podem converter-se em defeitos. Quando mecanismos existentes de tolerância a falhas percebem o erro, estes podem atuar com o intuito de conter a propagação do mesmo. Se estas ações tiverem sucesso, ocorre a recuperação do sistema, caso contrário, este encontra-se com defeito.

2.2. Falhas em Barramentos

De acordo com Zimmer & Jantsch [10] a tecnologia *submicron* terá de lidar com mais falhas transientes do que antes. Basicamente, o número de *Single Event Upsets* (SEUs) devidos as mais diversas razões (injeção de carga de nêutrons, ou partículas alfa, variações no processo, interferência eletromagnética) aumentarão juntamente com *crosstalk* (acoplamento capacitivo ou indutivo entre linhas de transmissão de um barramento), afetando principalmente interconexões longas [11]. A miniaturização da tecnologia também trouxe a tona novas fontes de erros: os efeitos de nêutrons (que anteriormente eram somente uma preocupação relacionada a memórias) agora podem causar falhas em elementos lógicos e barramentos de baixa potência/baixa impedância [12]. No entanto, a principal causa de erros em barramentos é o *crosstalk*, que pode ocasionar atrasos no sinal ou causar erros nas linhas destes barramentos [13]. Além disso, *crosstalk* pode inerentemente causar erros bidirecionais ou em múltiplos *bits* (i.e. distúrbios em múltiplas linhas, causando defeitos nas transições $0 \rightarrow 1$ e $1 \rightarrow 0$) [14]. Falhas em barramentos maciçamente paralelos entre interconexões adjacentes são majoritariamente transientes e não necessariamente confinadas a uma única linha do barramento, portanto modelos de falhas convencionais não são suficientes para descrevê-los com precisão.

2.3. Modelos de Falhas em Barramentos

Birner e Handl [32] dividem os modelos em barramentos falhas em três camadas: *camada física, camada lógica* e *camada funcional*.

2.3.1. Camada Física

A camada física é entendida como a abstração que envolve os defeitos físicos do sistema em questão. Esta camada é subdividida em duas classes: uma para as interconexões da placa de circuitos e uma para as conexões internas ao circuito integrado.

A maioria dos defeitos encontrados nesta camada são causados pelo processo de fabricação. Problemas comuns são curto-circuitos e circuitos abertos causados por mperfeições nas etapas de corrosão e deposição, tanto da placa de circuitos quanto do circuito integrado. Outras falhas são causadas pela eletromigração de metais, devido a altas densidades de corrente e estresse elétrico, como descargas eletrostáticas.

2.3.2. Camada Lógica

O uso de uma abstração de mais alto nível permite a classificação de diversos modos de falhas físicas em um modelo de falhas lógico, tornando possível a injeção de falhas sem a manipulação do sistema físico. A Figura 2.2 mostra a o modelo de falhas em três camadas e os submodelos da camada lógica.

Modelo Geométrico

O representante mais famoso do nível geométrico, embora possa ser visto como um modelo de chaveamento ou estrutural, é o modelo de falhas *bridging*. Falhas de *bridging* são causadas por todos os curto-circuitos existentes entre linhas que, normalmente, não são conectadas. Linhas conectadas indevidamente podem assumir diferentes valores, dependendo da

tecnologia envolvida. Existem diversos modelos de falhas de bridging para descrever cada caso individualmente. No caso mais simples, estas falhas são modeladas como funções OR ou AND.

Este submodelo cobre, principalmente, defeitos gerados no processo de fabricação. Devido às geometrias cada vez menores, estas falhas são cada vez mais comuns. De acordo com [33], entre 40% e 50% de todas as falhas podem ser modeladas com o modelo *bridging*.

Modelo Estrutural

Esta abstração envolve portas lógicas AND, OR, NAND, NOR e XOR cujas interconexões sejam defeituosas. É esperado que o funcionamento da porta lógica em si seja correto. O modelo de falhas mais utilizado deste nível é o *stuck-at-1* ou *stuck-at-0* (quando uma linha tem seu valor fixo em nível lógico alto ou baixo). De acordo com [34] entre 80% e 85% de todos os defeitos físicos podem ser detectados com este modelo.

Modelo de Atraso

Um defeito pode afetar tanto o comportamento lógico de um circuito quanto seu comportamento no tempo. Um exemplo é o aumento do tempo de propagação do de um sinal, que pode resultar na leitura incorreta do mesmo.

Modelo de Chaveamento

O modelo de chaveamento trata de falhas internas aos transistores de um circuito. Um exemplo de modelo de falhas nesse nível é o modelo *stuck-open*, que ocorre quando um dos transistores do circuito perde a capacidade de transferir carga e apresenta a saída em alta impedância.

Modelo de *Crosstalk*

Crosstalk é um fenômeno pelo qual um sinal cria um efeito indesejado em outra parte do circuito sem que haja uma causa material. Existem três tipos de *crosstalk*: resistivo, indutivo e capacitivo. Internamente aos circuitos integrados, devido às suas pequenas dimensões, o *crosstalk* capacitivo é o que gera maiores efeitos. Em uma placa de circuitos, o *crosstalk* manifesta-se de forma magnética, ou indutiva, devido às correntes maiores envolvidas. Normalmente, um modelo de falhas *crosstalk* expressa-se na forma de falhas transientes ou de atrasos de propagação.

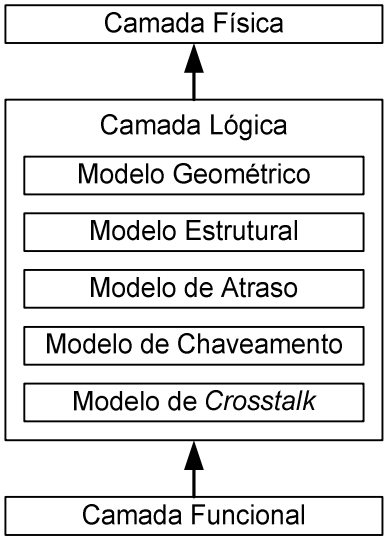

Figura 2.2 – Modelos de falhas divididos em camadas [32].

2.3.3. Camada Funcional

Os modelos de falhas a nível de portas lógicas provêm descrições acuradas de defeitos físicos em potencial. Estes modelos, no entanto, muitas vezes, são complexos demais. Desta forma, é necessário incluir tantos defeitos quanto possível dentro de um modelo funcional. Infelizmente não existem modelos gerais, de forma que soluções específicas devem ser desenvolvidas caso a caso.

2.4. Notação de Modelos de Falhas em Barramentos Utilizada

Zimmer e Jantsch afirmam em [10] que tentativas anteriores de descrever modelos de falhas em barramentos, utilizando modelos de falhas clássicos como *bridging* e *stuck-at*, não foram capazes de fornecer uma notação completa de todos os tipos de falhas esperados. Uma nova notação para modelos de falhas foi, então, desenvolvida, de forma a gerar modelos que explicitamente suportem falhas compreendendo múltiplas linhas, múltiplos ciclos em barramentos implementados em uma ou mais camadas de um circuito integrado ou placa de circuito impresso.

No presente trabalho, para auxiliar na implementação e utilização da plataforma de injeção de falhas proposta, é utilizada uma simplificação da notação desenvolvida em [10]. Nesta simplificação, o modelo de falhas é descrito pelo seu efeito, pela sua característica temporal e por quais linhas são afetadas pelo mesmo.

2.4.1. Efeito

O efeito de um modelo de falha descreve qual alteração uma linha de um barramento sofrerá no momento da ativação da falha. A Tabela 2.1 mostra uma lista incompleta de efeitos:

Tabela 2.1 – Representação dos efeitos das falhas

	Efeito *e*	Descrição
1	*Inv*	O sinal da linha é invertido
2	*Set0*	Linha forçada para 0
3	*Set1*	Linha forçada para 1
4	*SetRand*	Linha forçada para um valor aleatório
5	*Bridge*	Linha w_i é forçada para o valor da linha w_{i-1}
6	*Del*	Linha é forçada para seu valor anterior
...	...	...

2.4.2. Característica Temporal

A característica temporal de uma falha descreve o intervalo de tempo no qual uma falha está ativa, alterando o barramento onde ocorre, de acordo com a Tabela 2.2:

Tabela 2.2 – Característica temporal de um modelo de falha.

	Característica	Descrição
1	Transiente	A falha gera o efeito por um único ciclo de *clock.*
2	Transiente Longa	A falha gera o efeito por mais de um ciclo de *clock.*
3	Intermitente	A falha possui um período ativo, quando o efeito é gerado e um período inativo, quando o efeito não age sobre o barramento
4	Permanente	A falha está presente desde o momento da ativação do sistema até o final do experimento.

2.4.3. Linhas afetadas

As linhas de um barramento afetadas pelo modelo de falhas em questão são indicadas simplesmente como uma máscara binária a ser aplicada sobre o barramento no momento da ativação da falha.

2.5. Falhas em Memórias

2.5.1. Modelo Funcional de Memória

O modelo funcional de uma memória normalmente consiste em muitos blocos, cada um representando uma função específica, conforme a Figura 2.3 Cada um desses blocos pode vir a apresentar defeitos, entretanto, falhas em diferentes blocos podem vir a apresentar o mesmo comportamento. Levando-se em conta essa característica, é possível adotar um modelo simplificado de memória para fins de modelamento de falhas, mostrado na Figura 2.4 [4][5][6].

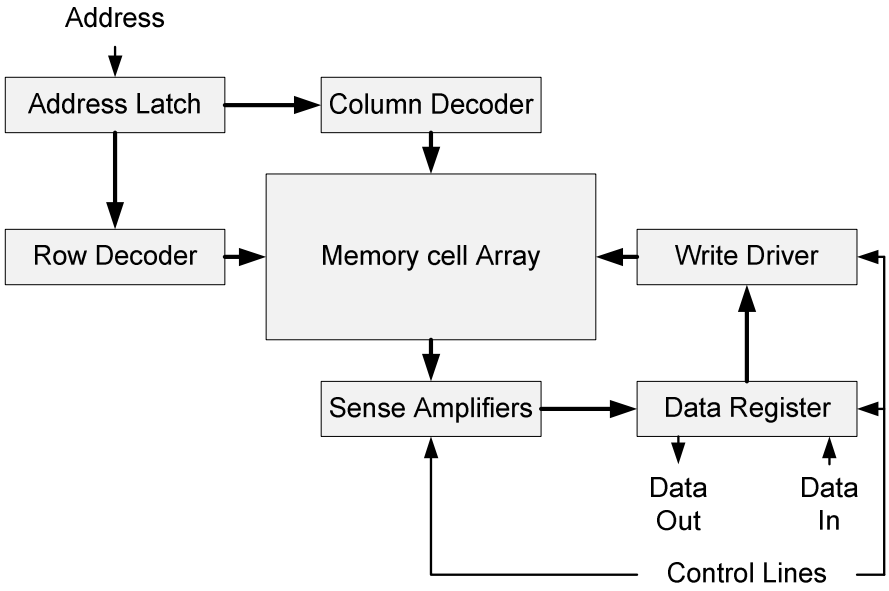

Figura 2.3 – Modelo funcional de uma memória SRAM [4].

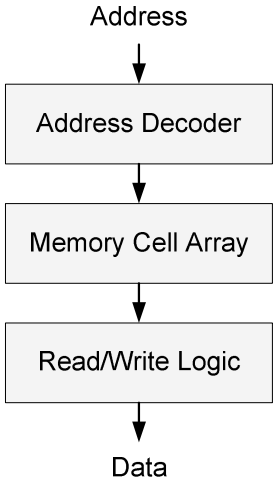

Figura 2.4 – Modelo simplificado de memória [4].

A seguir, descreve-se os principais blocos funcionais que compõe uma memória RAM, bem como os tipos mais comuns de falhas que podem ocorrer nestes blocos.

A) Lógica de Leitura/Escrita (*read/write logic*)

A lógica de leitura/escrita passa as informações dos pinos para a célula de memória e vice-versa. De acordo com [5], falhas nos barramentos, amplificadores e registradores resultam nas seguintes tipos de falhas:

- Um ou mais *bits* do barramento estão colados em um valor (*stuck-at*);
- Um ou mais *bits* do barramento estão abertos (*stuck-open*);
- Um par ou mais *bits* estão acoplados (*coupling*).

As falhas na lógica de leitura/escrita podem ser vistas como falhas na matriz de memória [5]. O primeiro item é equivalente a m conjunto de falhas *stuck-at*. O segundo item é equivalente a um conjunto de falhas *stuck-open*. Finalmente, o terceiro item é equivale a um conjunto de células acopladas em um mesmo endereço.

B) Decodificador de Endereços (*address decoder*)

O decodificador de endereços é simplesmente um circuito lógico combinacional que seleciona uma única célula de memória para cada endereço diferente. De acordo com [5], as falhas possíveis no decodificador de endereço são:

- Mais de uma célula é acessada por um mesmo endereço;
- Um endereço não acessa nenhuma célula.

Falhas no decodificador também podem ser vistas como falhas na matriz de memória. O primeiro caso é equivalente a um acoplamento direto (*direct coupling*) e o segundo item é equivalente a uma célula isolada (*stuck-open*) [5].

C) Matriz de células de memória (*memory cell array*)

Baseando-se nos defeitos físicos mais comuns em matrizes de memória, como curtos na metalização e acoplamento capacitivo, são descritos os seguintes tipos de falhas [4][5][6]:

- Uma ou mais células estão coladas em ‘0’ ou ‘1’;
- A célula não pode ser acessada;
- A célula é incapaz de efetuar uma transição de um valor x para um valor y;
- A célula não é capaz de manter o seu estado ao longo do tempo;
- Existe um ou mais pares de células que apresentam algum tipo de acoplamento. Por exemplo, uma transição de x para y em uma célula *i* implica na transição de x para y em uma célula *j*, sem que, necessariamente, o inverso seja verdadeiro;

Estas falhas dão origem aos modelos de falhas clássicos que serão apresentados com mais detalhes na seção seguinte.

2.6. Modelos de falhas em memórias

2.6.1. Comportamento correto

Para fins de comparação, a Figura 2.5 mostra o diagrama de Markov de uma célula de memória que apresenta um comportamento correto. A notação utilizada nas figuras seguintes é definida como:

- S0, S1: Estado da célula, ‘0’ ou ‘1’, respectivamente;
- W0, W1: Operação de escrita de um valor ‘0’ ou ‘1’, respectivamente;
- R0, R1: Operação de leitura na célula, resultando um valor ‘0’ ou ‘1’, respectivamente.

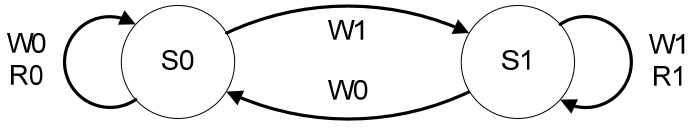

Figura 2.5 – Diagrama de Markov para uma célula de memória saudável [7]

2.6.2. Stuck-At

Este modelo de falha define que o estado de uma célula de memória está fixado em ‘0’ ou ‘1’, ou seja, independente da operação que seja realizada sobre a mesma, o seu estado não se altera [7]. A Figura 2.6 mostra o diagrama de Markov para o comportamento de uma célula que apresenta uma falha *stuck-at-0.*

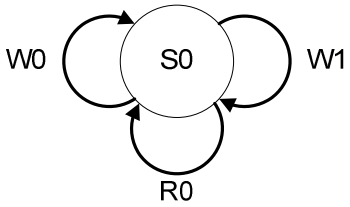

Figura 2.6 – Diagrama de Markov para um modelo de falha *stuck-at-0*. [7]

2.6.3. Stuck-Open

Uma falha tipo *stuck-open* ocorre quando uma célula não pode ser acessada. Quando uma operação de leitura é realizada, o valor lido vai depender da implementação dos amplificadores de saída e pode ser caracterizada como uma falha *stuck-at-0* ou *stuck-at-1* [8].

2.6.4. State Coupling

Este modelo de falha define que o estado de uma célula *j* é forçado para um determinado valor somente se uma célula *i* está em determinado estado, sem que sejam realizadas nenhuma operação em qualquer uma destas células [9].

Na Figura 2.7, é mostrado o diagrama de Markov de um conjunto de duas células, *i* e *j*, que estão sujeitas a um *state coupling*. Neste exemplo, o valor da célula *j* é fixado em ‘0’ sempre que a célula *i* também possuir o valor ‘0’.

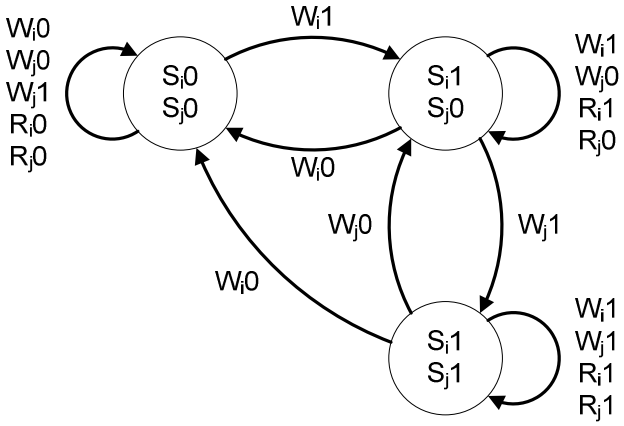

Figura 2.7 – Diagrama de Markov para um modelo de falha *state-coupling*.

2.7. Conclusão

A implementação de uma plataforma de injeção de falhas necessariamente passa pelo estudo dos modelos de falhas que esta plataforma será capaz de realizar. Este estudo permite a identificação do comportamento das falhas representadas por estes modelos bem como seu mecanismo de propagação e a forma como estas falhas são percebidas pelo usuário do sistema.

O uso de uma notação para a construção de modelos de falhas específicos para barramentos, separando os efeitos das falhas no barramento de seu comportamento temporal, permitiu que a plataforma de injeção de falhas fosse desenvolvida com uma arquitetura modular.

3. Técnicas de Injeção de Falhas

3.1. Introdução

Conforme os estudos realizados por [15], injeção de falhas é um conjunto de técnicas e ferramentas utilizadas para avaliar a confiabilidade e segurança de um sistema. Esta é uma abordagem experimental que consiste na introdução deliberada de falhas no sistema - o sistema alvo - e na observação de seu comportamento sob tais condições. No contexto de sistemas tolerantes a falhas, engenheiros utilizam a injeção de falhas para avaliar a atuação de mecanismos de tolerância a falhas de um sistema ou de seus componentes. Durante estes experimentos são avaliadas a capacidade de detecção e de isolamento de falhas bem como as capacidades de reconfiguração e recuperação. Pode-se dizer que a natureza muitas vezes destrutiva de defeitos e as latências de erro muito grandes encontradas em campo levaram ao desenvolvimento de tais técnicas visando ao aumento da ocorrência de falhas.

Diferentes técnicas de injeção de falhas são utilizadas em diferentes fases do desenvolvimento de um projeto, da conceituação até o produto final. Estas técnicas podem ser divididas em técnicas baseadas em simulação, em *hardware* ou em *software*.

Ferramentas baseadas em simulação são utilizadas para a avaliação da confiabilidade de um sistema nas fases de conceituação e projeto. Neste ponto, o sistema em questão é apenas uma série de abstrações de alto nível e detalhes de implementação ainda não foram determinados. Desta forma, o sistema é simulado com base em suposições simplificadas. A injeção de falhas baseada em simulações, que assume que erros ou defeitos ocorrem de acordo com uma distribuição predeterminada, provê respostas rápidas para os engenheiros do sistema. No entanto, estas técnicas requerem parâmetros de entrada precisos, que são difíceis de fornecer. Mudanças no projeto e troca de tecnologias muitas vezes inviabilizam o uso e medições feitas anteriormente.

Experimentos realizados em protótipos, no entanto, permitem a avaliação do sistema sem nenhuma suposição sobre o projeto, o que gera resultados mais precisos. Falhas são injetadas em protótipos para:

- Identificar gargalos de confiabilidade;
- Estudar o sistema na presença de falhas;
- Determinar a cobertura de mecanismos de detecção e recuperação de erros;
- Perda de desempenho.

Em protótipos, a injeção de falhas ocorre ao nível de *hardware* (falhas lógicas ou elétricas) ou ao nível de *software* (corrupção do código ou dos dados) e seus efeitos são monitorados. O sistema utilizado para a avaliação pode ser tanto um protótipo quanto um exemplar finalizado do sistema. A injeção de falhas em um sistema operacional pode revelar informações sobre o processo de defeito. No entanto, a injeção de falhas é adequada para estudar somente falhas emuladas. Esta abordagem não é capaz de gerar métricas de confiabilidade, como tempo médio entre falhas e disponibilidade.

Ao invés de injetar falhas, projetistas também podem medir diretamente os sistemas conforme eles lidam com a carga de trabalho real. A análise baseada em mensuração usa dados reais, que contêm muita informação sobre erros e defeitos que ocorrem naturalmente, assim como tentativas de recuperação. Ao analisar estes dados, é possível entender as características reais de erros e defeitos. Esta abordagem tem a desvantagem de ser limitada apenas aos erros detectados, que precisam ser coletados por um longo período de tempo, porque erros e defeitos ocorrem infreqüentemente. As condições do ambiente no qual o sistema está inserido também podem variar, comprometendo o valor estatístico destes resultados.

3.2. Arquitetura

A Figura 3.1 mostra o diagrama de blocos de um ambiente de injeção de falhas, que tipicamente consiste em:

- Controlador: Gerencia o experimento. Pode ser uma plataforma independente ou fazer parte do sistema alvo.
- Gerador de carga de trabalho: Fornece os comandos e tarefas a serem executados pelo sistema alvo. A carga de trabalho é obtida de uma biblioteca associada e pode ser composta de aplicações, benchmarks ou tarefas sintéticas, desenhadas para exercitar partes específicas do sistema.
- Monitor: Faz a interface entre o controlador e o sistema alvo, observa o sistema em teste e transporta os dados obtidos para o coletor de dados.
- Injetor de falhas: Hardware ou software responsável por introduzir falhas no sistema. Idealmente é capaz de injetar diversos tipos de falhas (variando a localização, temporização e estrutura do sistema alvo), obtidas da biblioteca de falhas.
- Sistema alvo: Sistema sob teste.
- Analisador de dados: Processa os dados obtidos durante as campanhas de injeção de falhas. Pode atuar em tempo real ou ser um processo separado.

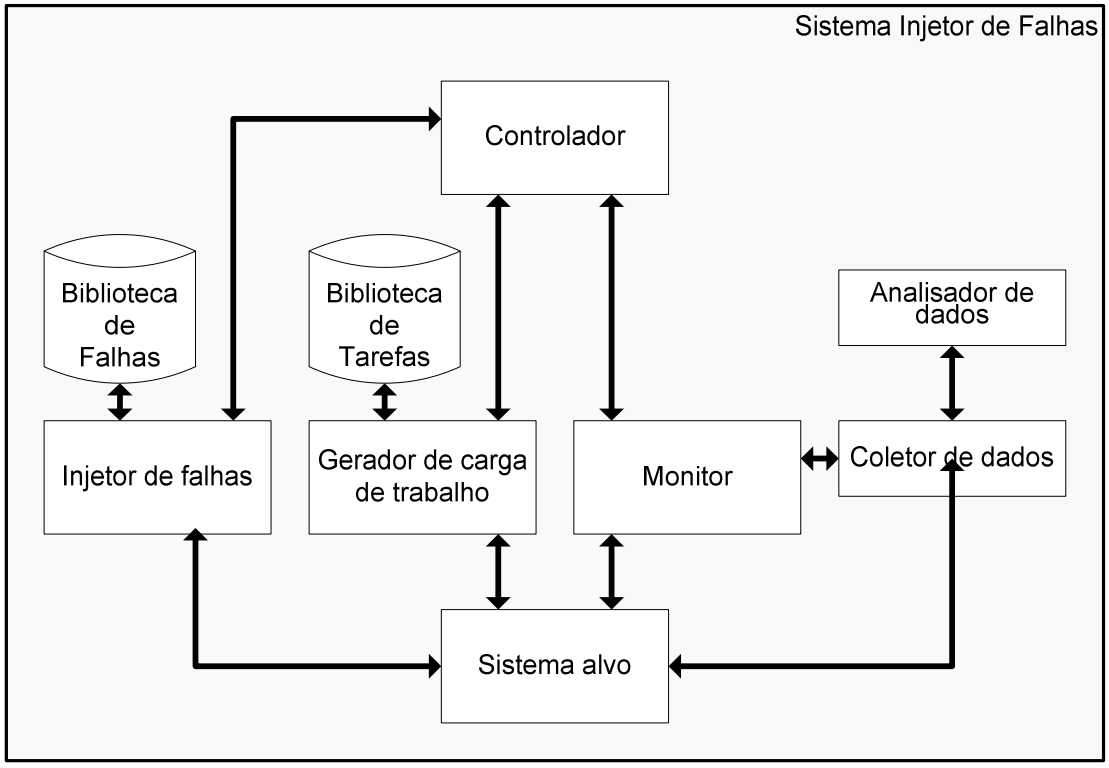

Figura 3.1 – Arquitetura de um ambiente de injeção de falhas 15.

3.3. Atributos

A literatura [16][17][15] utiliza os seguintes atributos para descrever as características de uma técnica de injeção de falhas:

- Acessibilidade: Habilidade de acessar determinados pontos de um sistema com a finalidade de gerar falhas. E.g. Pinos, registradores, células de memória;
- Controlabilidade: O grau de controle que o usuário tem ao indicar quando e onde as falhas devem ser injetadas;
- Custo: O custo de um experimento. Levam-se em consideração os recursos, financeiros ou não, decorrentes da construção e de cada rodada do experimento;
- Intrusão: O quanto o ambiente de injeção de falhas afasta o sistema alvo de suas condições de operação normal. E.g. Dado um sistema de tempo real, um injetor de falhas que interrompe sua execução para executar a injetar as falhas é mais intrusivo que um injetor que executa suas funções em tempo real;
- Portabilidade: O quanto é necessário modificar um injetor de falhas para que o mesmo possa atuar em um novo sistema alvo. Quanto menor a necessidade de alterações, maior é a portabilidade;
- Repetibilidade: Refere-se a habilidade de reproduzir resultados estatísticos ou idênticos para um ambiente de testes. Uma alta repetibilidade gera resultados iguais ou semelhantes, garantindo uma alta confiabilidade nos experimentos;
- Resolução temporal: A velocidade de aquisição das informações monitoradas, tanto informações da saída do sistema como estados internos que permitam a geração de falhas;
- Risco: O risco de danificar o sistema alvo no processo de injeção de falhas.

3.4. Técnicas de Injeção de Falhas por Simulação

Nas técnicas de injeção de falhas por simulação, um modelo do sistema sob teste, o qual pode ser desenvolvido em diferentes níveis de abstração é simulado em outro sistema

computacional [18]. Falhas são introduzidas alterando os valores lógicos dos elementos do modelo durante a simulação. Na fase de projeto do sistema, a simulação é uma forma de avaliar o desempenho e a confiabilidade de um sistema. Outra vantagem interessante desta técnica em respeito a outras técnicas de injeção é a alta observabilidade e controlabilidade de todos os componentes modelados.

Particularmente, existe um grupo de ferramentas de injeção de falhas baseadas no uso da linguagem VHDL como linguagem para desenvolvimento de modelos. Estas técnicas são largamente aplicadas devido as vantagens de empregar uma linguagem de descrição de *hardware* padronizada. Essas Técnicas podem ser classificadas nas seguintes categorias [18]:

- Comandos de simulador: Estas técnicas baseiam-se no uso de comandos do simulador durante a simulação para modificar os valores de sinais e variáveis sem a necessidade de alteração do código VHDL;
- Modificação no código VHDL: Nesta abordagem o código VHDL que descreve os modelos do sistema é alterado utilizando sabotadores, modificando os componentes do modelo ou utilizando extensões da linguagem.

Infelizmente, técnicas de injeção de falhas baseadas em simulação geralmente requerem um esforço computacional extremo, especialmente se um nível de detalhes muito grande é desejado. Uma forma de limitar este esforço é confinar a simulação a intervalos muito curtos de tempo real, o que torna difícil observar os efeitos em longo prazo de uma falha injetada.

A) Comandos de simulador

A principal razão para utilizar os comandos internos de um simulador para injeção de falhas é que isto não requer modificações no código VHDL. No entanto, a aplicabilidade destas técnicas depende das funcionalidades oferecidas pelas linguagens dos simuladores.

Duas técnicas baseadas no uso de comandos de simulador foram identificadas e utilizadas: a corrupção de construtos *signal* e *variable* que caracterizam respectivamente as funcionalidades estruturais e comportamentais de um modelo descrito em VHDL [18]. A forma

como estas falhas são injetadas dependem do ponto de injeção. Para injetar falhas em um *signal*, a seqüência de pseudo-comandos utilizada é:

1. Simular_Ate(instante da injeção);
2. Modificar_Sinal(nome do sinal, valor);
3. Simular_Por(duração da falha);
4. Restaurar_Sinal(nome do sinal);
5. Simular_Por(tempo de observação);

Esta seqüência é utilizada para injetar falhas transientes, as quais são as mais comuns e difíceis de detectar. Para injetar falhas permanentes, a seqüência é a mesma, apenas omitindo os passos 3 e 4. Para injetar falhas intermitentes, a seqüência de comandos consiste em repetir os passos 1 a 5 com intervalos aleatórios.

Para injetar falhas em um constructo *variable*, é utilizada a seguinte seqüência:

1. Simular_Ate (instante da injeção);
2. Alterar_Variavel(nome da variável, valor);
3. Simular_Por(tempo de obseervação);

Esta operação é similar a injeção em sinais, mas neste caso não existe controle da duração da falha. Isso implica que não é possível injetar falhas permanentes em variáveis utilizando comandos de simulador.

B) Modificação no código VHDL

Nesta categoria, duas técnicas podem ser aplicadas. A primeira é baseada na adição de componentes dedicados à injeção de falhas como *sondas*, ou *sabotadores*. A segunda é baseada na *mutação* de descrições de componentes existentes do modelo VHDL. Estas técnicas são aplicadas respectivamente aos modelos estruturais e comportamentais do modelo alvo.

Benso apresenta uma classificação de sabotadores [18]:

- Sabotador Serial Simples: Interrompe a conexão entre uma saída e sua respectiva entrada, modificando o valor de recepção;
- Sabotador Serial Simples Bi-direcional: Possui dois sinais de entrada/saída, mais um sinal de leitura/escrita que determina a direção da perturbação;
- Sabotador Serial Complexo Interrompe a conexão entre duas saídas e seus respectivos receptores, modificando o valor da recepção;
- Sabotador Serial Complexo Bi-direcional: Possui quatro sinais de entrada/saída, mais um sinal de leitura/escrita que determina a direção da perturbação;
- Sabotador *n*-bits Unidirecional Simples: É usado em barramentos unidirecionais de *n* bits (endereço e controle). É composto de *n* sabotadores seriais simples;
- Sabotador *n*-bits Bi-direcional Simples: É usado em barramentos bi-direcionais de *n* bits (endereço e controle). É composto de *n* sabotadores seriais bi-direcionais simples;
- Sabotador *n*-bits Unidirecional Complexo: É usado em barramentos unidirecionais de *n* bits (endereço e controle). É composto de *n/2* sabotadores seriais complexos;
- Sabotador *n*-bits Bi-direcional Complexo: É usado em barramentos bi-direcionais de *n* bits (endereço e controle). É composto de *n/2* sabotadores bi-direcionais complexos;

Os sinais de controle ativam a injeção. Estes podem ser controlados através de comandos de simulador e sua ativação determina tanto o instante quanto a duração da injeção. Sinais de leitura/escrita determinam a direção da injeção das falhas.

Os sinais onde os sabotadores podem ser inseridos são aqueles que conectam componentes em modelos estruturais. A arquitetura interna dos sabotadores pode ser estrutural ou comportamental. O projeto comportamental é basicamente um processo cuja lista de sensitividade contém os sinais de controle e os sinais de entrada/saída. O projeto estrutural é baseado no uso de multiplexadores.

A desvantagem do uso de sabotadores é que um número de sinais de controle deve ser adicionado ao modelo. Estes sinais são utilizados para ativar um entre um conjunto de sabotadores inseridos e para indicar o tipo de perturbação que este irá inserir. Isso adiciona complexidade tanto ao modelo quanto à técnica. Sabotadores são mais difíceis de implementar que comandos de simulador, porém estes podem utilizar um conjunto maior de modelos de falhas,

De acordo com [18] um mutante é um componente que substitui outro componente. Enquanto inativo, este funciona da mesma forma que o componente original, mas, quando ativado, funciona como aquele na presença de falhas.

A mutação pode ser implementada de diferentes formas:

- Adicionando um ou mais sabotadores às descrições estruturais ou comportamentais dos componentes;
- Modificando a descrição estrutural substituindo sub-componentes. Por exemplo, uma porta NAND pode ser substituída por uma porta NOR;
- Modificando as estruturas sintáticas da descrição comportamental.

3.4.1. MEFISTO

MEFISTO (*Multi-level Error/Fault Injection Simulation TOol*) [19] foi criado em conjunto pelo LAAS-CNRS, em Toulouse, França e o LDS de Chalmers, Gothenburg, Suécia. MEFISTO é um ambiente integrado e coerente de projeto de sistemas tolerantes a falhas que utiliza a linguagem VHDL para descrever modelos e realizar experimentos de injeção de falhas via simulação.

MEFISTO pode ser usado para (i) estimar a cobertura de mecanismos tolerantes a falhas; (ii) investigar diferentes mecanismos para mapear resultados de um nível de abstração para outros e (iii) avaliar modelos de falhas e erros aplicados durante experimentos de injeção de falhas conduzidos em uma implementação de um sistema tolerante a falhas. Esta ferramenta utiliza comandos de simulador para controlar tanto mutantes quanto sabotadores.

3.5. Técnicas de Injeção de Falhas em Hardware

Técnicas de injeção de falhas em *hardware* consistem em gerar falhas físicas no *hardware* do sistema real utilizando *hardware* adicional, interno ou externo ao sistema. Esta é uma abordagem alternativa a injeção de falhas em simulação, que costumam ser bastante complexas a ponto de não serem capazes de validar completamente o sistema. [20]. Outra vantagem é a aproximação dos modelos de falhas utilizados com os modelos de falhas reais.

As técnicas de injeção de falhas em *hardware* podem ser divididas em duas categorias:

- **Com contato:** O injetor possui contato físico direto com o sistema alvo, produzindo alterações de tensão ou corrente externamente aos circuitos integrados do sistema. Exemplos: métodos que utilizam sondas e/ou soquetes.
- **Sem contato:** O injetor não possui contato físico direto com o sistema alvo. As falhas são injetadas através de fenômenos físicos produzidos por uma fonte. Exemplos: métodos que utilizam fontes radioativas ou interferência eletromagnética para induzir corrente nas estruturas internas dos circuitos integrados do sistema alvo.

Estes métodos são adequados para estudar as características de confiabilidade de protótipos que requerem uma alta resolução temporal para o acionamento e monitoração (e.g: latência de falhas na CPU) ou requerem acesso a locais que não podem ser afetados por outros métodos de injeção de falhas. Geralmente são implementados modelos de falhas de baixo nível - uma falha *bridging* pode ser um curto circuito. A alta resolução temporal é atingida através do *hardware* que introduz a falha e também monitora seu impacto. Normalmente, o injetor aplica a falha após um período de tempo pré-determinado, controlado via um temporizador em *hardware* ou após detectar um evento, tal como um padrão específico em um barramento de endereços.

A) Injeção com contato

O método mais comum de injeção de falhas via *hardware* é através do contato direto com pinos do circuito [15]. Existem duas técnicas principais capazes de alterarem as correntes e tensões elétricas nestes pinos:

- **Sondas ativas:** Esta técnica injeta corrente via sondas ligadas aos pinos, alterando as correntes presentes no funcionamento normal do sistema. O método das sondas é, normalmente, limitado a modelos de falha *stuck-at*, embora seja possível ligar duas ou mais sondas para emular falhas *bridging*. O projetista deve tomar cuidados especiais para que a corrente inserida pela sonda não danifique o *hardware* do sistema alvo.

- **Soquete:** Nesta técnica é inserido um soquete entre o circuito integrado alvo e a placa de circuitos impressos. O soquete pode inserir falhas tipo *stuck-at*, stuck-*open*, ou alguma lógica mais complexa ao forçar, entre os pinos do sistema, sinais analógicos que representam os níveis lógicos desejados. Os sinais nestes pinos podem ser alterados de acordo com sinais presentes em outros pinos ou mesmo com valores anteriores do mesmo pino.

Ambos os métodos são capazes de um alto grau de controle sobre os locais e instantes nos quais as falhas são aplicadas. É importante notar que estas falhas, por serem modeladas ao nível dos pinos do sistema, são diferentes das encontradas dentro dos circuitos integrados.

Além das falhas em pinos e barramentos, outra abordagem bastante utilizada é a injeção de falhas nos barramentos do circuito de alimentação do sistema alvo através de sondas conectadas à fonte de alimentação.

B) Injeção sem contato

Estas falhas são injetadas ao expor o sistema a fenômenos físicos como campos eletromagnéticos ou radiação ionizante. A falha ocorre pelas correntes geradas através de indução eletromagnética ou pelos íons passando através da zona de depleção do circuito integrado. Estes métodos são bastante utilizados porque simulam eventos naturais encontrados no ambiente real de operação do sistema. A desvantagem é que é difícil controlar exatamente o momento e o local da injeção da falha porque não é possível controlar exatamente o momento da emissão de um íon ou criação de um campo eletromagnético.

3.5.1. Messaline

Messaline [21] foi desenvolvida no LAAS-CNRS, em Toulouse, França. Esta técnica utiliza tanto sondas ativas quanto soquetes para conduzir a injeção de falha no nível dos pinos do sistema. A Figura 3.2 mostra a arquitetura geral deste ambiente.

Messaline pode injetar falhas tipo *stuck-at*, circuito aberto, curto circuito e falhas lógicas complexas, entre outras. É possível, também, controlar a duração de cada falha e sua freqüência Além disso, um dispositivo é associado a cada ponto de injeção para monitorar se e quando uma falha é ativada e produz um erro. É possível injetar falhas em até 32 pontos simultaneamente.

Esta ferramenta foi utilizada em experimentos de um sistema de controle centralizado de ferrovias e em sistemas distribuídos para o projeto Delta-4.

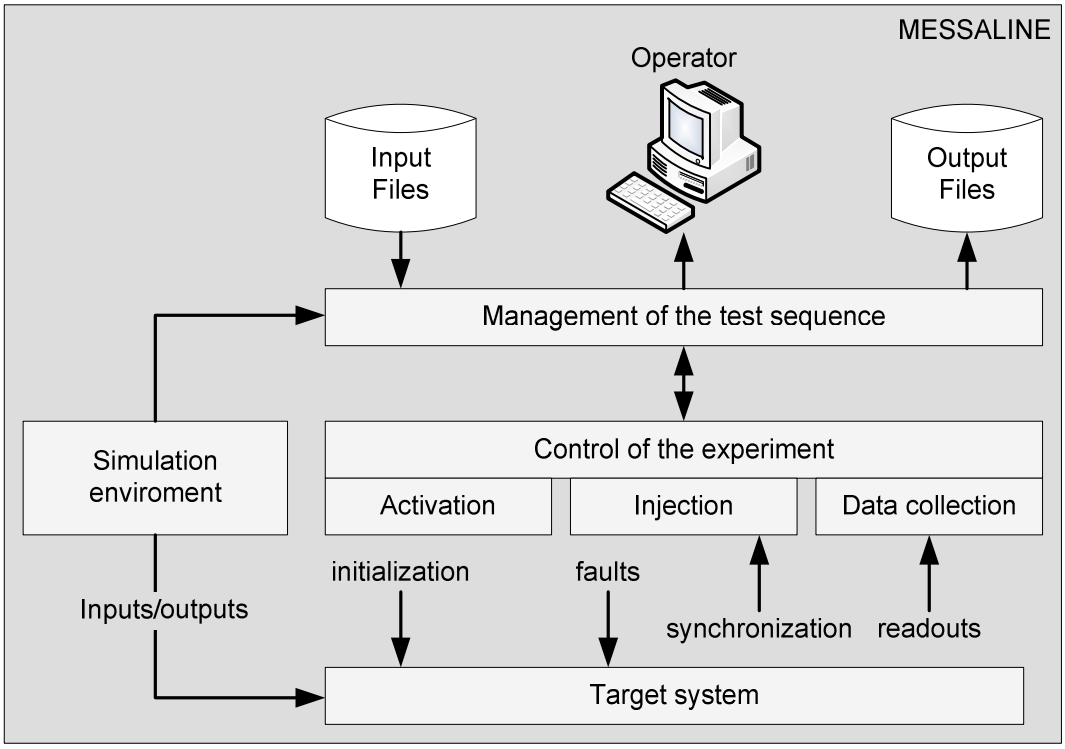

Figura 3.2 – Arquitetura geral do ambiente Messaline [21].

3.5.2. FIST

FIST [22] (*Fault Injection for Study of Transient fault effect*) foi desenvolvido na universidade de tecnologia de CHALMERS na Suécia e emprega métodos com contato e sem contato para criar falhas transientes dentro do sistema alvo. Esta ferramenta utiliza radiação de íons pesados para criar falhas transientes em pontos aleatórios dentro de um circuito integrado quando este é exposto à radiação e pode então causar inversões de bits simples ou múltiplas. A fonte de radiação é montada dentro de uma câmara de vácuo juntamente com um pequeno computador de dois processadores. O computador é posicionado de forma que um dos processadores seja exposto diretamente abaixo da fonte de radiação. O outro processador é usado como uma referência para detectar se a radiação resulta na inversão de algum bit. A Figura 3.3 mostra o ambiente FIST.

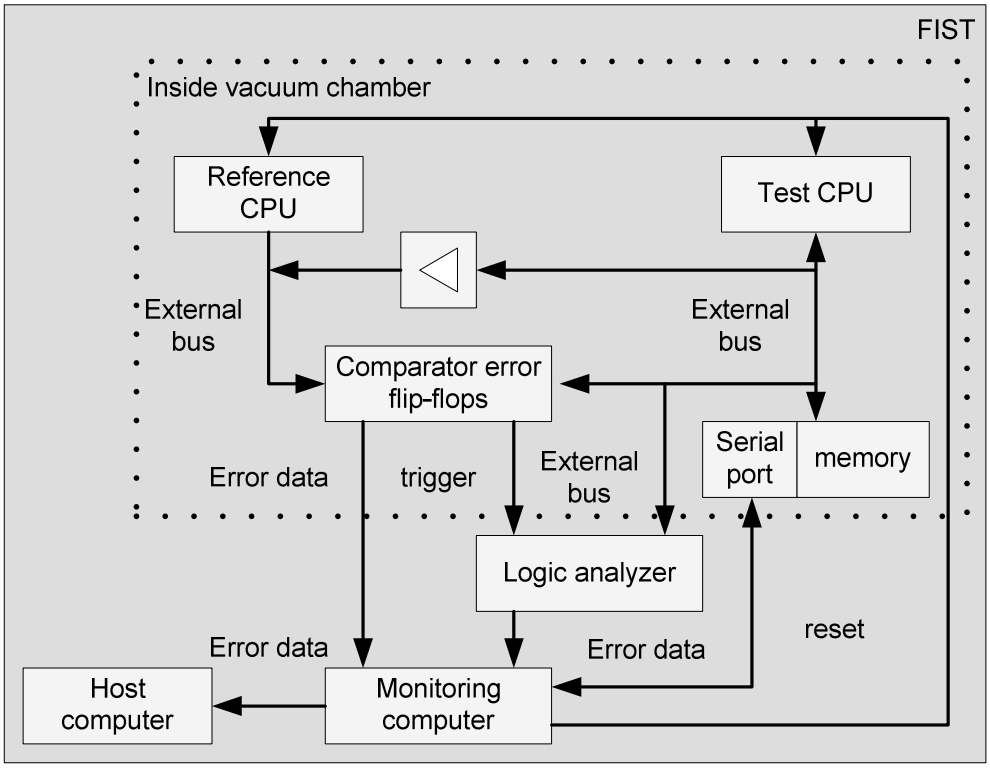

Figura 3.3 – Arquitetura do ambiente FIST [22].

A ferramenta FIST pode injetar falhas diretamente no interior de um circuito integrado, o que não pode ser feito com injeções em pinos externos. Esta ferramenta pode produzir falhas transientes em pontos aleatórios do circuito integrado, o que leva a uma grande variação de erros vistos nos pinos de saída. Além disso, a ferramenta FIST permite a injeção de distúrbios nas linhas de alimentação. Isso é realizado utilizando um transistor MOS entre a fonte de alimentação e o pino Vcc do circuito integrado do processador para controlar a amplitude da queda de tensão. Distúrbios na fonte de alimentação geralmente afetam pontos múltiplos dentro do circuito integrado e podem causar erros de atraso de propagação em sinais internos. Os resultados experimentais mostram que os erros resultantes de ambos os métodos causam efeitos similares no controle de fluxo de programa e erros de dados. No entanto a radiação de íons pesados, na maioria das vezes, causa falhas nos barramentos de endereços, enquanto distúrbios na fonte de alimentação afetam majoritariamente sinais de controle.

3.5.3. FITO

FITO [23] (*FPGA-based fault Injection TOol*) foi desenvolvida no DSL da Universidade de Tecnologia Sharif, no Irã. Esta ferramenta baseia-se na injeção de falhas em modelos descritos em Verilog sintetizáveis. FITO suporta modelos de falhas permanentes e transientes utilizando modificações no código Verilog do sistema alvo. A descrição do circuito é sintetizável e pode ser utilizada para experimentos de injeção de falhas em FPGAs. A Figura 3.4 mostra a arquitetura da ferramenta.

Para suportar modelos de falhas permanentes, a ferramenta utiliza a inserção de portas lógicas em sinais específicos do sistema alvo. Para modelos de falhas transitórias, as mesmas portas lógicas são inseridas, mas uma de suas entradas é ligada a um pequeno núcleo controlador que gerencia a injeção da falha baseado em temporizadores. Estes temporizadores são configurados pelo usuário através de uma ferramenta que automatiza o processo de modificação do código.

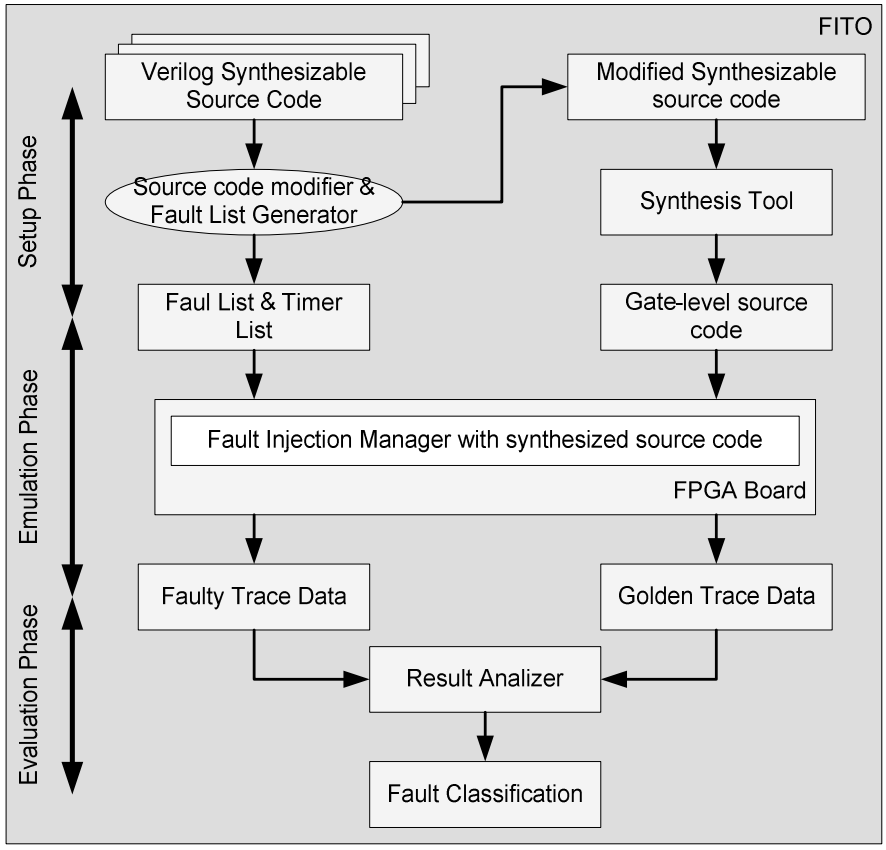

Figura 3.4 – Arquitetura da Ferramenta FITO [23].

3.5.4. ARROW

ARROW (A Generic Hardware Fault Injection Tool for Nocs) [32] foi desenvolvido pelo Instituto de Engenharia de Computação da Universidade de Tecnologia de Viena, Áustria. Esta ferramenta utiliza sabotadores para injetar falhas em interconexões de NoCs (Network-on-Chip) descritos em VHDL.

Esta ferramenta pode injetar falhas a nível lógico ou de atraso de propagação nas linhas das interconexões existentes entre diversos blocos de uma rede intra chip. É possível injetar falhas cuja duração seja menor que um ciclo de *clock* utilizando de um divisor de freqüência e cadeias de *flip-flops*. A ferramenta é configurada e operada através de uma porta serial RS232 ligada a um computador pessoal.

3.6. Técnicas de Injeção de Falhas em Software

Recentemente, pesquisadores passaram a se interessar em desenvolver ferramentas de injeção de falhas baseadas em software. Essas técnicas de injeção de falhas baseadas em *software* representam soluções extremamente viáveis no que diz respeito ao custo associado as mesmas, visto que não requerem equipamentos caros. Além disso, elas podem ser usadas para interagir com aplicações e sistemas operacionais, o que é difícil de realizar com técnicas de injeção de falhas baseadas em *hardware*. Se o alvo é uma aplicação, o injetor de falhas é inserido na própria aplicação ou entre esta e o sistema operacional. Se o alvo é o sistema operacional o injetor de falhas deve ser embarcado no próprio sistema operacional, pois é extremamente difícil criar uma camada intermediária entre a máquina e o sistema operacional.

Apesar de flexível, a abordagem de injeção de falhas via *software* possui algumas desvantagens:

- Não pode ser usada para inserir falhas em locais inacessíveis ao *software*;
- A instrumentação do *software* pode interferir com a carga de trabalho que está rodando no sistema alvo e mesmo mudar a estrutura do software original. Assim, um cuidado especial deve ser tomado para projetar um ambiente de injeção que possa minimizar essa perturbação, quando se tratam de aplicações em tempo real.

Tais técnicas também apresentam uma resolução temporal pobre, o que pode causar problemas de fidelidade em relação ao sistema não-modificado. Para falhas de grande latência, tais como falhas em memória, a baixa resolução temporal não chega a ser um problema, enquanto que para falhas de curta latência, como falhas em barramentos e no processador, esta abordagem pode deixar de capturar algum comportamento errôneo. Projetistas podem resolver estes problemas ao adotar uma abordagem híbrida, combinando a versatilidade do software com a precisão da monitoração via hardware. Abordagens híbridas também são adequadas para medir latências extremamente pequenas. No entanto, o hardware envolvido pode custar mais e diminuir a flexibilidade ao limitar os pontos de observação e a capacidade do armazenamento de dados.

É possível categorizar os métodos de injeção de falhas em software com base no momento no qual as falhas são injetadas: durante o tempo de compilação ou durante o tempo de execução.

A) Injeção em tempo de compilação

Para injetar falhas em tempo de compilação, o conjunto de instruções do programa deve ser modificado antes que a imagem do programa seja carregada e executada. Ao invés de injetar falhas no hardware do sistema alvo, este método injeta erros no código fonte ou no código objeto do programa alvo para emular o efeito de falhas no hardware, no software e falhas transientes. Quando o sistema executa a imagem alterada, as falhas são ativadas. Este método requer a modificação do programa que vai avaliar o efeito das falhas, mas não necessita de software adicional. Também não são causadas perturbações no sistema alvo. Como os efeitos das falhas são constantes, podem ser usados para modelar falhas permanentes. Este método é bastante simples de ser implementados, mas não permite a injeção de falhas durante o tempo de execução.

B) Injeção em tempo de execução

Durante o funcionamento do sistema, é necessário um mecanismo para ativar a injeção de falhas. Mecanismos comuns incluem:

- **Temporizador:** O mecanismo mais simples, um temporizador expira após um intervalo predeterminado, gerando uma interrupção que invoca a injeção. O temporizador pode ser implementado em *software* ou *hardware* e deve estar ligado ao mecanismo de tratamento de interrupções do sistema. Este método não requer modificações na carga de trabalho do programa. Como esta técnica injeta falhas com base no tempo, ao invés de eventos específicos do sistema, são produzidos efeitos imprevisíveis no comportamento da aplicação. Este comportamento é adequado para simular falhas transientes e intermitentes.

- **Exceção:** Neste caso, uma exceção de hardware ou de software transfere o controle da aplicação para o injetor de falhas. Diferentemente do mecanismo baseado em temporizadores, é possível injetar falhas sempre que um determinado evento ou condição ocorrer. Por exemplo uma exceção de software pode ativar a injeção de falhas sempre que uma instrução específica for ser executada. Quando uma exceção de *software* ou *hardware* é executada, é gerada uma interrupção e o controle da execução é transferido.

- **Inserção de código:** Nesta técnica, instruções que permitem a ocorrência da injeção são adicionadas ao programa alvo, de forma muito semelhante ao método de modificação de código. Mas, ao contrário deste, aquela executa a injeção em tempo de execução e adiciona instruções ao invés de alterar as instruções existentes. Diferentemente, do método que utiliza exceções, esta forma de injetor de falhas pode existir como parte da aplicação alvo e pode rodar em modo usuário.

3.6.1. Xception

Xception [31] foi desenvolvido na Universidade de Coimbra, em Portugal. Esta ferramenta utiliza as funcionalidades de depuração e monitoração de desempenho presentes em muitos processadores modernos para injetar falhas realistas. Não são necessárias modificações no software e não é necessária a inserção de nenhuma exceção de *software*. A ferramenta utiliza o próprio *hardware* que controla as exceções para injetar as falhas. A injeção de falhas é

implementada como um gerenciador de exceções e requer a modificação do vetor de interrupções. As falhas são ativadas baseadas no acesso a endereços específicos (ao invés do período de tempo após um evento), de forma que os experimentos são reprodutíveis. Os seguintes eventos podem ativar a injeção de falhas:

- Busca de *opcode* de um endereço específico;
- Carga de operando de um endereço específico;
- Armazenamento de um operando em um endereço específico;
- Intervalo desde a inicialização;
- Uma combinação dos eventos acima.

Cada falha possui uma máscara de falhas específica: um conjunto de bits determina quais bits correspondentes na localização alvo serão injetados. Bits na máscara de falha setados para '1' podem usar várias operações: *stuck-at-zero*, *stuck-at-one*, *bit-flip* e *bridging*.

Xception foi implementada numa máquina paralela Parsytec, baseada no processador PowerPC 601. Experimentos revelaram deficiências nos mecanismos de detecção de erros ao mostrar que até 72% das falhas injetadas resultaram em resultados incorretos que não foram detectadas por algumas unidades funcionais do processador.

3.6.2. DOCTOR

Doctor (IntegrateD SOftware Fault InjeCTiOn EnviRonment) [24], desenvolvido na universidade de Michigan permite a injeção de falhas na CPU, falhas de memória e falhas de comunicação por rede. Este ambiente utiliza três métodos de ativação (limite de tempo, exceção e modificação de código) para ativar a injeção de falhas. O limite de tempo ativa a injeção de falhas em memória. Uma vez que este limite de tempo é atingido, o injetor de falhas é acionado para sobrescrever uma região de memória emulando uma falha. Para falhas não permanentes na CPU, exceções ativam a injeção de falhas. Para falhas permanentes, a injeção de falhas é realizada modificando instruções do programa durante a compilação, emulando corrupção de dados e instruções. Doctor foi utilizada em Harts, um sistema distribuído de tempo real para investigar o efeito de perdas intermitentes de mensagens entre dois nós adjacentes e o efeito do

roteamento. Pesquisadores utilizaram resultados experimentais para validar um modelo de entrega de mensagens e avaliar diferentes métodos de entrega de mensagens.

3.7. Conclusão

Neste capítulo foram apresentadas as principais abordagens para a realização de experimentos de injeção de falhas em sistemas computacionais. Técnicas baseadas em simulação envolvem a descrição e a manipulação de modelos abstratos do sistema a ser avaliado. Desta forma é possível observar e injetar falhas em todos os pontos do sistema. Estes modelos, no entanto crescem em complexidade conforme descrevem mais e mais aspectos do sistema original, exigindo um esforço computacional elevado para a realização de um experimento que, mesmo assim, não fornece completa certeza de que o modelo final, no mundo físico, se comportará da mesma maneira. Técnicas baseadas em *hardware*, por outro lado, utilizam implementações físicas do sistema que está sendo avaliado, e, portanto, fornecem resultados mais realistas. A desvantagem desta abordagem é o uso de sistemas adicionais acoplados ao sistema-alvo, possivelmente alterando seu comportamento em relação ao comportamento original. Finalmente, técnicas de injeção de falhas em software são atraentes, pois não exigem modificações físicas no sistema a ser avaliado, mas, acabam por alterar, mesmo que sutilmente seu comportamento, no momento da injeção da falha, inviabilizando seu uso em algumas aplicações, como sistemas em tempo real.

4. Plataforma de Injeção de Falhas Proposta

4.1. Introdução

A ferramenta de injeção de falhas proposta foi desenvolvida visando obter a maior flexibilidade possível em termos de acesso ao sistema alvo. A nova plataforma explora algumas características relacionadas às abordagens baseadas em simulação e às abordagens baseadas em *hardware*, o que permite que a mesma seja implementada em um sistema físico, de forma a permitir experimentos em situações mais próximas às reais condições de operação do sistema alvo. A nova plataforma foi implementada utilizando a linguagem VHDL (*Very High Speed Iintegrated Circuit Hardware Description Language*), e FPGAs (*Field Programmable Gate Array*).

Esta técnica objetiva injetar falhas em barramentos de SoCs. Considerando que falhas injetadas em barramentos conectados a memórias podem ser mapeadas para falhas na própria memória, é dito que a ferramenta proposta é capaz de injetar falhas tanto em barramentos como em memórias, seguindo os modelos de falhas descritos no Capítulo 2. Por exemplo, uma falha *stuck-at-0* em uma célula de memória pode ser realizada com um efeito *Set0*, de duração permanente, ativo em apenas um endereço do barramento de dados conectado a esta memória.

O processo de injeção de falhas é baseado no uso de sabotadores aplicados a sinais do sistema alvo. Essa abordagem, derivada das técnicas baseadas em simulação, permite a injeção de falhas de forma transparente, sem alterar o funcionamento do sistema alvo. É importante salientar que esta técnica é minimamente intrusiva, pois apenas um elemento lógico é adicionado ao caminho combinacional dos sinais funcionais alvos. Como resultado, o atraso adicional para todo o circuito é desprezível. A contribuição deste trabalho está no método de controle flexível e na criação de um ambiente que suporta esta abordagem.

O ambiente de injeção de falhas é composto de duas partes. A primeira é um software que é executado pelo computador pessoal, denominado Módulo Controlador. Este módulo gerencia a execução das campanhas de injeção de falhas e controla a aquisição de dados do sistema gerados durante os experimentos. A segunda parte consiste na plataforma de hardware, onde o sistema alvo e o Módulo Injetor estão instanciados em um FPGA. O Módulo Injetor, por sua vez é composto de duas partes, o Núcleo de Controle e o Núcleo Injetor. O Núcleo de Controle é um microcontrolador que executa, utilizando o Módulo Injetor, um algoritmo que descreve o modelo de falhas considerado durante o experimento de injeção de falhas de acordo com parâmetros definidos pelo usuário. O Núcleo Injetor é instanciado como um periférico do núcleo de controle e é responsável pela manifestação física dos efeitos do modelo de falhas no sistema alvo.

A Figura 4.1 mostra o esquema básico do sistema:

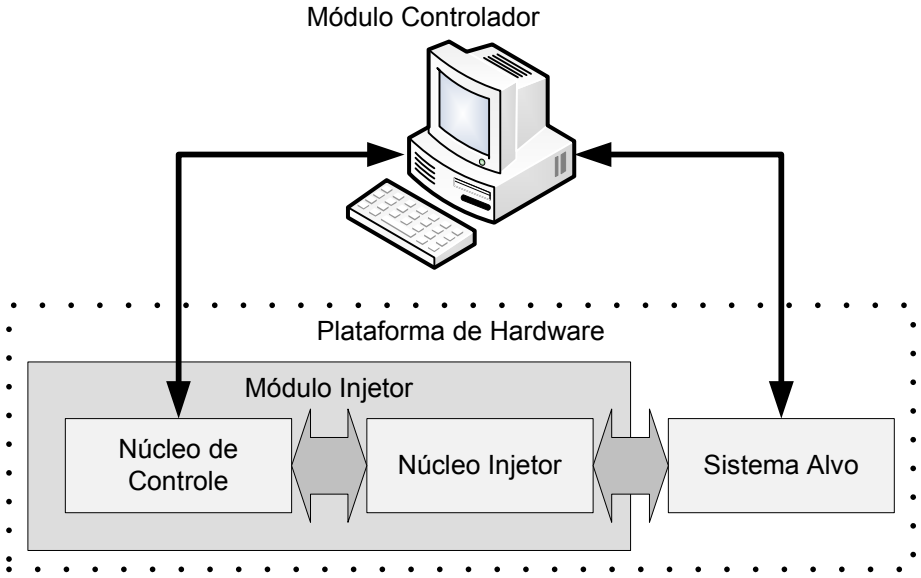

Figura 4.1 – Ambiente para injeção de falhas proposto.

Na Figura 4.2 é possível observar que, logicamente, o Núcleo injetor é colocado sobre um determinado barramento, entre dois componentes do sistema alvo, com o intuito de injetar uma determinada falha de acordo com o modelo especificado pelo usuário.

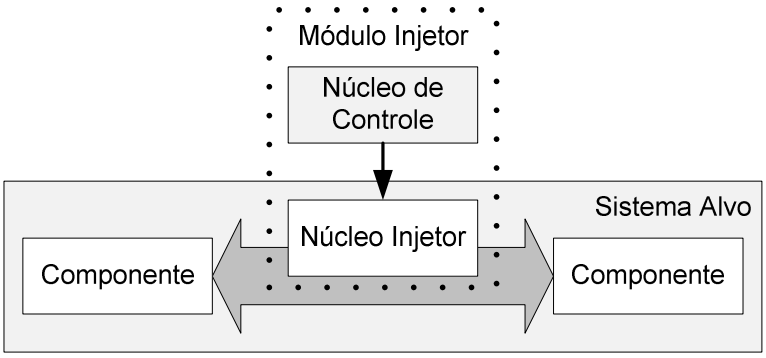

Figura 4.2 – Interface entre módulo injetor e barramento do sistema alvo.

Nas próximas seções serão descritos os componentes do ambiente proposto e será apresentado um roteiro detalhado para a criação de um experimento e injeção de falhas.

4.2. Plataforma de Hardware

Este ambiente foi desenvolvido para ser utilizado em placas de desenvolvimento de FPGAs comuns. Especificamente, foram utilizados FPGAs e ferramentas de síntese, configuração e compiladores Xilinx. É importante salientar que o FPGA deve ter capacidade suficiente para conter o sistema alvo mais o Módulo Injetor e dois pinos de entrada/saída para a comunicação serial.

A técnica é voltada para sistemas microcontrolados, e espera uma comunicação via porta serial com o sistema alvo. Neste caso é necessário reservar mais um par de pinos de entrada/saída. Caso seja de interesse utilizar esta técnica com um circuito que não possua uma porta serial, é possível adaptar um núcleo que executar essa comunicação.

4.2.1. Sistema Alvo

Qualquer sistema descrito em VHDL sintetizável pode ser utilizado como sistema alvo. Este sistema deve ser modificado para que seja possível realizar a interface com o Módulo Injetor. Essa modificação consiste na exposição do sinal onde será injetada a falha no nível superior da hierarquia da descrição do sistema.

No exemplo abaixo, o objetivo é injetar uma falha entre os sinais *fonte* e *destino* de um componente descrito em VHDL. Para tal, exteriorizam-se os sinais através da criação de novas portas no componente.

```
destino <= fonte;
```

Deve ser convertido para:

```
Y : in  std_logic;
X : out std_logic;
...
X <= fonte;
destino <= Y;
```

Onde X representa o sinal original e Y representa o sinal sujeito a falhas. Estes sinais, X e Y, serão conectados ao Núcleo Injetor da plataforma de injeção de falhas.

4.2.2. Módulo Injetor

Conforme visto nteriormente, o Módulo Injetor é composto de dois blocos funcionais denominados Núcleo de controle e Núcleo Injetor. Estes dois blocos serão descritos em detalhes nos próximos parágrafos.

Núcleo de Controle

O Núcleo de Controle é responsável pela configuração e controle do Núcleo Injetor de acordo com parâmetros recebidos do Módulo Controlador e do Algoritmo de Injeção de Falhas executado. Este núcleo consiste basicamente de um processador *Microblaze,* desenvolvido pela Xilinx [25]. O *Microblaze* é um processador *soft-core* – implementado em lógica sintetizável – com arquitetura Harvard RISC (*Reduced Instruction Set Computer*, Computador com conjunto reduzido de instruções), otimizado para FPGAs Xilinx. Este processador específico foi escolhido simplesmente pela facilidade de uso e pela facilidade de criação e conexão de periféricos.

Alguns periféricos estão conectados ao *Microblaze,* como um gerador de números aleatórios, um contador/temporizador e uma porta serial para comunicação com o controlador.

Algoritmo de Injeção de Falhas

Chamamos de Algoritmo de Injeção de Falhas a aplicação escrita em linguagem C executada pelo Módulo de Controle durante o experimento de injeção de falhas. Esta aplicação compreende o recebimento de parâmetros definidos pelo usuário e a execução de um algoritmo que descreva o modelo de falhas adotado durante um experimento utilizando o Núcleo Injetor para gerar os efeitos físicos associados ao modelo. Foi criada uma pequena biblioteca de funções para a interface com o Núcleo Injetor.

Exemplo 1: Um modelo de falhas *stuck-at-x* em um barramento qualquer pode ser escrito, em pseudocódigo como:

```
1.  serial_Recebe(TIPO);
2.  serial_Recebe(MASCARA);
3.  alvo_MantemReset();
4.  injetor_ConfiguraTemporizador(PERMANENTE)
5.  injetor_ConfiguraTipo(TIPO);
6.  injetor_ConfiguraMascara(MASCARA);
7.  alvo_LiberaReset();
8.  espera();
```

Neste algoritmo o Núcleo de Controle recebe do controlador dois parâmetros, TIPO e MASCARA. O parâmetro TIPO informa qual o tipo de alteração que será realizada no barramento enquanto o parâmetro MASCARA indica quais linhas do barramento serão afetadas. O Núcleo de Controle, então, segura o sistema alvo em uma situação de *reset* enquanto configura o Núcleo Injetor. Terminada a configuração, o sinal de *reset* é liberado. Como a falha é permanente, o Módulo de Controle não executa mais nenhuma instrução e mantém o barramento em *stuck-at* até o final do experimento.

Exemplo 2: Um modelo de falha intermitente, onde o efeito da falha é aplicado periódicamente no sistema alvo, pode ser descrito da seguinte forma:

```
1.  serial_Recebe(TIPO);
```

```
2.  serial_Recebe(MASCARA);
3.  serial_Recebe(TEMPO_ON);
4.  serial_Recebe(TEMPO_OFF);
5.  alvo_MantemReset();
6.  injetor_ConfiguraMascara(MASCARA);
7.  injetor_ConfiguraTipo(TIPO);
8.  injetor_ConfiguraIntervaloAtivo(TEMPO_ON);
9.  injetor_ConfiguraIntervaloLimpo(TEMPO_OFF);
10. espera();
```

Neste exemplo, temos novamente a recepção dos parâmetros vindos do Módulo Controlador e a configuração do Núcleo Injetor enquanto o sistema alvo fica paralisado em uma situação de *reset*. Dessa vez é utilizado o modo cíclico do Núcleo Injetor, que ativa a falha pelo tempo TEMPO_ON e desativa pelo tempo TEMPO_OFF, ciclicamente até ser desabilitado.

Núcleo Injetor

O Núcleo Injetor é responsável por fazer a interface entre o Núcleo de Controle e sinais do sistema alvo bem como a geração da falha propriamente dita. A Figura 4.3 apresenta o diagrama de blocos do Núcleo Injetor, composto de quatro blocos funcionais mais o sistema alvo.

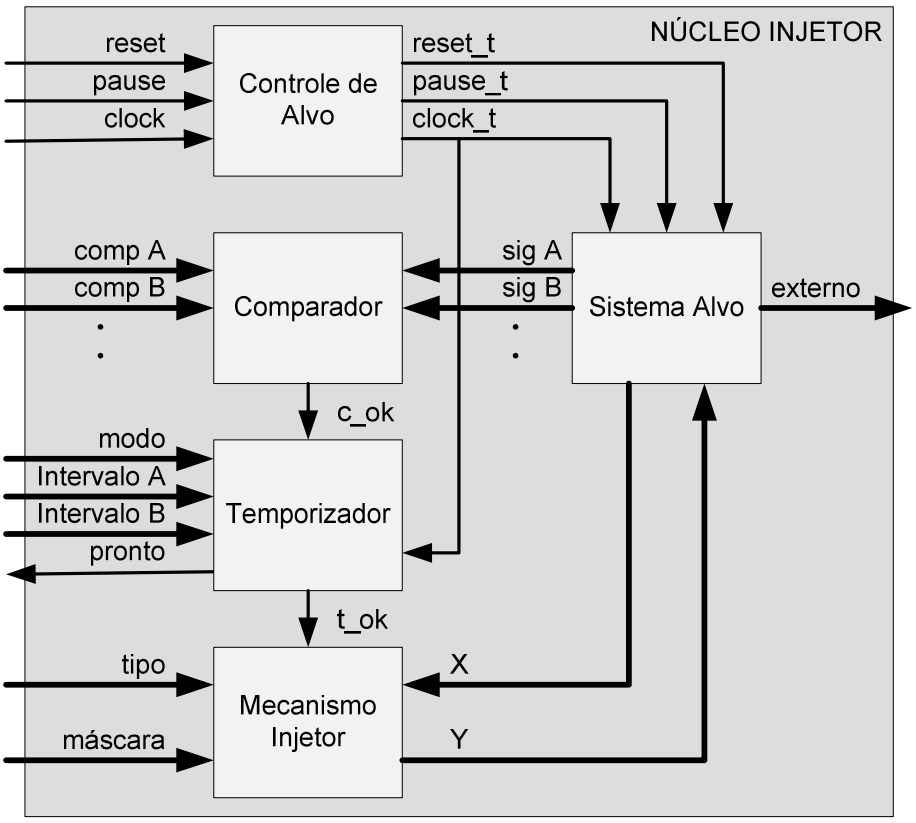

Figura 4.3 – Arquitetura interna do Núcleo Injetor.

As condições a seguir devem estar presentes para que a injeção de uma falha seja executada:

- O sistema alvo deve estar em um estado válido para a injeção de falhas. Um estado válido para injeção de falhas é definido como o conjunto de estados dos componentes internos do sistema alvo pertinentes para o experimento no momento no qual deve ser executada a injeção. Por exemplo, se um experimento consiste em injetar falhas em uma região da memória de um SoC, um estado válido ocorre quando o barramento de endereços apontar para um endereço dentro dessa região e o sinal de controle de leitura/escrita estiver indicando uma escrita. Este estado deve ser definido pelo usuário durante o planejamento do experimento.

- O momento deve ser correto de acordo com o modelo de falhas em questão.

Assim, o Bloco Comparador monitora os estados do sistema alvo enquanto o Bloco Temporizador é responsável pela execução da característica temporal do modelo de falhas. Quando essas duas condições forem cumpridas o Mecanismo Injetor é habilitado para realizar a injeção da falha.

O Núcleo Injetor é instanciado como um periférico do Núcleo de Controle e ligado a um de seus barramentos. O sistema alvo, por sua vez, é instanciado como um bloco dentro do Núcleo Injetor.

A) Bloco Controle de Alvo

O bloco Controle de Alvo possui duas funções: fornecer o sinal de *clock* para o sistema alvo e expor os sinais de *reset* e, se presente, *pause* para o Núcleo de Controle.

O sinal de *clock* para o sistema alvo é gerado a partir da divisão ou não do *clock* do Núcleo de Controle, de acordo com as necessidades do sistema alvo. A utilização de uma fonte de *clock* comum é importante para garantir a sincronia de operação entre ambos os sistemas.

B) Bloco Comparador

O bloco Comparador é responsável por analisar certos sinais do sistema alvo e compará-los com os parâmetros recebidos do Núcleo de Controle. Esta comparação tem por objetivo verificar se o sistema alvo está em um estado válido para a injeção de falhas. Caso a comparação seja positiva, o sinal “c_ok” é posto em nível lógico ‘1’, caso contrário em nível lógico ‘0’.

Este bloco é o único que não possui uma implementação padronizada, pois cada sistema alvo possui um conjunto de sinais diferentes de onde é possível extrair informações sobre seu estado interno. Além disso, cada experimento pode apresentar uma definição diferente para o que é um estado válido para injeção de falhas.

Por exemplo, no caso de um experimento de injeção de falhas no barramento de escrita da memória de um processador, os sinais a serem monitorados são o barramento de endereços e

os sinais de controle do barramento de dados. Assim, a comparação procuraria por uma faixa de endereços válida para a memória e por sinais de controle que indicassem uma escrita (*chip select*, *write enable*, etc).

C) Bloco Temporizador

O bloco Temporizador é responsável pela temporização do efeito das falha a ser aplicada. Este bloco possui três sinais de entrada: “modo”, “intervalo” e “c_ok”, e dois sinais de saída: “pronto” e “t_ok”. Suas funções estão descritas na tabela abaixo:

Tabela 4.1 – Sinais do bloco Temporizador.

Sinal	Direção	Largura	Função
modo	Entrada	3 bits	Configura o modo de operação do bloco. 000: Desabilitado – Nenhuma falha é injetada. 001: Permanente – Falha permanente. 010: Falha Transiente – Falha com duração de um ciclo de *clock*. Aguarda c_ok == ‘1’ para ativar a injeção. 011: Falha Transiente Longa – Libera injeção pela duração especificada no sinal “intervalo A”; 100: Intermitente – Neste modo o mecanismo injetor é habilitado por “intervalo A” ciclos e desabilitado por “intervalo B” ciclos. A máquina de estados permanece nesse estado até que outro modo seja configurado.
intervalo A	Entrada	32 bits	Configura o intervalo de duração de uma falha transiente longa. Após este intervalo a injeção de falhas é desabilitada. E a máquina de estados volta para o estado desabilitado.
intervalo B	Entrada	32 bits	Configura o intervalo no qual o mecanismo de injeção é desabilitado quando no modo cíclico.
c_ok	Entrada	1 bit	Comparador OK 0: Sistema alvo está em estado inválido para injeção; 1: Sistema alvo está em estado válido para injeção.
pronto	Saída	1 bit	0: Falha transiente ou falha transiente longa está sendo injetada; 1: Fim da injeção de falha transiente ou falha transiente longa, falha permanente ou estado desabilitado.
t_ok	Saída	1 bit	0: Desabilita mecanismo injetor 1: Habilita mecanismo injetor.

O controle da temporização é implementado como uma máquina de estados, como mostra a Figura 4.4:

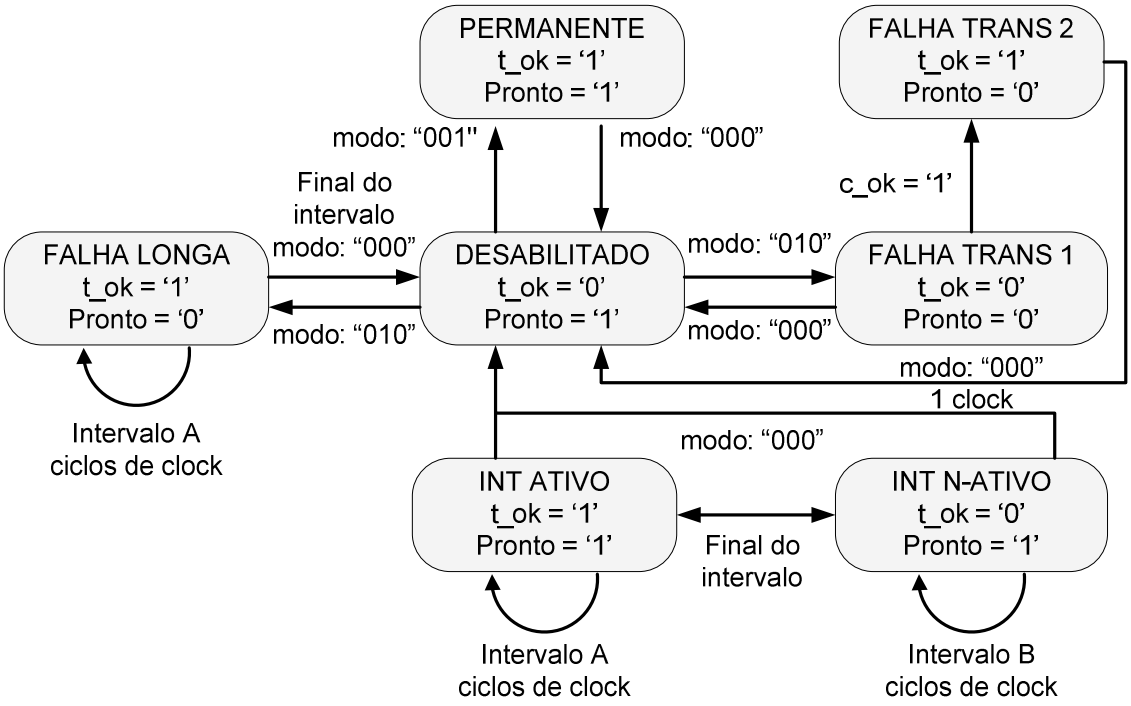

Figura 4.4 – Diagrama de estados do bloco temporizador.

O sinal "t_ok", como definido na tabela acima, indica uma situação válida para a execução da injeção de uma falha e habilita o mecanismo injetor. Este sinal é setado quando o bloco comparador indicar uma condição válida, "c_ok" igual a '1', e a máquina de estados do bloco temporizador estiver em um estado apropriado, "Permanente", "Falha Longa" ou "Falha Curta 2".

É importante salientar que uma Falha Transiente percorre estados diferentes de uma Falha Transiente Longa. A configuração para o modo de Falha Transiente Longa libera a injeção de falhas imediatamente, enquanto uma Falha Transiente aguarda um sinal positivo do bloco Comparador. Caso o usuário deseje a execução imediata de uma Falha Transiente, pode-se utilizar o modo Falha Transiente Longa com o intervalo configurado para 1 ciclo de *clock*.

D) Bloco Mecanismo Injetor

Este bloco altera um sinal do sistema alvo produzindo o efeito físico da falha. Basicamente, o Mecanismo Injetor é implementado como um sabotador paralelo simples, capaz de realizar operações lógicas em um sinal ou barramento do sistema alvo. Um cuidado especial foi tomado para que este bloco fosse realizado da forma mais simples possível, a fim de eliminar eventuais atrasos de propagação em relação ao sinal original.

Existem três maneiras de alterar um determinado sinal lógico: podemos forçá-lo para ‘0’, para ‘1’, ou podemos inverter o seu valor. A forma mais simples de fazer isto é modificar o caminho do sinal inserindo uma porta lógica simples e um sinal extra. A Figura 4.5 mostra as três alternativas para alteração de um sinal original ‘X’. Quando o sinal ‘F’ estiver em nível lógico ‘1’, a alteração tem efeito. O sinal ‘Y’ é o sinal extra que recebe o sinal alterado de ‘X’. ‘X’ deve ser substituído por ‘Y’ em todos os locais onde este sinal é consumido.

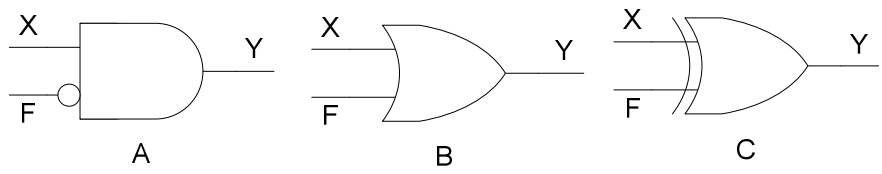

Figura 4.5 – Inclusão de portas lógicas para alterar um sinal lógico. A) Força o sinal para ‘0’, B) Força o sinal para ‘1’ e C) Inverte o sinal.

Nos FPGAs Xilinx, a lógica combinacional é realizada nos chamados recursos LUT (*Look-Up Table resources*). Estes recursos possuem, alguns circuitos auxiliares, como flip-flops para sincronização, multiplexadores para escolha de sinais de saída e uma LUT. Essas LUTs possuem quatro sinais de entrada e um sinal de saída. Qualquer função booleana de quatro variáveis pode ser implementada utilizando uma única LUT [26].

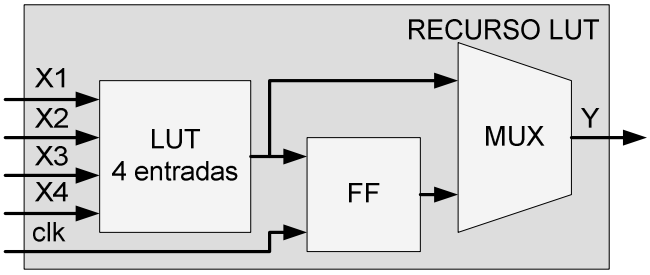

Figura 4.6 – Simplificação de um bloco tipo recurso LUT básico de um FPGA Xilinx [26].

O Mecanismo Injetor utilizado na técnica proposta utiliza uma LUT de quatro entradas para montar uma função que execute as três operações em um sinal discutidas anteriormente. Dessa forma, o atraso de propagação do sinal modificado é o menor possível para uma dada família de FPGAs e pode ser considerado desprezível a não ser em casos onde o sistema alvo já seja implementado no limite da tecnologia utilizada. A Figura 4.7 apresenta o bloco básico do Mecanismo Injetor. É importante notar a presença do sinal "t_ok", vindo do bloco temporizador, que habilita a injeção de falhas no mecanismo injetor.

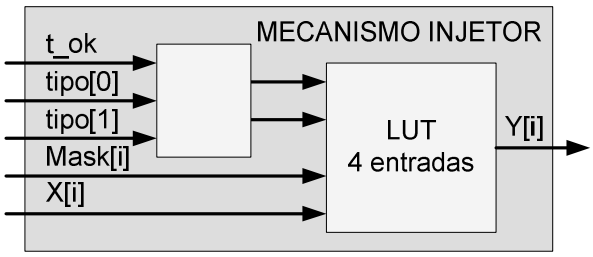

Figura 4.7 – Bloco básico do mecanismo injetor. É utilizado um bloco básico para cada linha X[i] de um barramento onde será inserida a falha.

As tabelas 4.2 e 4.3 descrevem os sinais do mecanismo injetor e sua tabela verdade, respectivamente.

Tabela 4.2 – Sinais do mecanismo injetor.

Sinal	Direção	Largura	Função
t_ok	Entrada	1 bit	0: Habilita o mecanismo injetor. 1: Y ← X

tipo	Entrada	2 bits	00: Mecanismo desabilitado. Y ← X; 01: *Set0*. Y ← 0; 10: *Set1*. Y ← 1; 11: *Inv*. Y ← ~X.
mask	Entrada	Até 32 bits	Máscara binária. 0: Os bits correspondentes de X não são afetados; 1: Os bits correspondentes de X são alterados pelo mecanismo injetor.
X	Entrada	Até 32 bits	Entrada do sinal original.
Y	Saída	Até 32 bits	Saída do sinal alterado.

Tabela 4.3 – Tabela verdade do bloco Mecanismo de Injeção.

	t_ok	tipo[0]	tipo[1]	mask	X	Y
Mecanismo de Injeção desabilitado	0	X	X	X	0	0
	0	X	X	X	1	1
Desabilitar	1	0	0	X	0	0
	1	0	0	X	1	1
Set0	1	0	1	0	0	0
	1	0	1	0	1	1
	1	0	1	1	X	0
Set1	1	1	0	0	0	0
	1	1	0	0	1	1
	1	1	0	1	X	1
Inv	1	1	1	0	0	0
	1	1	1	0	1	1
	1	1	1	1	0	1
	1	1	1	1	1	0

4.3. Módulo Controlador

O Módulo Controlador é um software desenvolvido em *Python* cuja responsabilidade é organizar e controlar a execução dos experimentos, configurar o FPGA na plataforma de *hardware*, transmitir o algoritmo de falhas e parâmetros de configuração para o Módulo de Controle e registrar os dados provenientes do Módulo de Controle e do sistema alvo.

4.3.1. Campanhas e Conjuntos de Campanhas

A unidade básica de um experimento de injeção de falhas é uma campanha. Definimos como campanha um conjunto de parâmetros que descrevem um determinado experimento em questão.

Esses parâmetros são definidos como:

- Nome: Um conjunto de caracteres que identifica o experimento;
- Repetições: Um número inteiro que indica quantas vezes a campanha deve ser repetida com os mesmos parâmetros;
- Tempo-limite: Um número inteiro que indica quantos segundos a campanha deve rodar, ao final do qual esta é interrompida;
- *Bitstream*: Um arquivo com a extensão *.bit* que contém a configuração a ser carregada no FPGA. Essa configuração deve conter o sistema alvo e o módulo injetor;
- Algoritmo de Injeção de Falhas: Um arquivo com a extensão *.elf* que contém a aplicação executada pelo núcleo de controle e descreve o modelo de falhas a ser aplicado no sistema alvo;
- Parâmetros diversos: São parâmetros descritos por um número hexadecimal de 32 bits a serem passados para o núcleo de controle após a configuração do FPGA. Estes parâmetros descrevem algumas características do modelo de falhas descrito no Algoritmo de Injeção de Falhas, como faixa de endereços onde a injeção de falhas deve ser executada, tempo de duração de falhas, repetições, etc. O número de parâmetros é limitado pelo Algoritmo de Injeção de Falhas. Todos os parâmetros devem estar presentes.

Um conjunto de campanhas é uma lista de uma ou mais campanhas descritas em um arquivo texto, com formato tipo *.csv* (valores separados por vírgulas). Os conjuntos de campanhas cumprem o papel de automatizar a execução dos experimentos. Um conjunto de campanhas é construído de acordo com o modelo a seguir:

```
# um ‘#’ na primeira posição indica um comentário
# ‘set’ na primeira posição indica um nome para o conjunto de campanhas
set, BFI Test Set 1
# Experimento, falha permanente
#nome, repetições, limite, bits, elf, inicio, fim, tipo, mascara
P1, 1, 10, bfi_test, permanent, 00001000, 00001400, 80000000, 000ff000
P2, 1, 10, bfi_test, permanent, 00001200, 00001600, c0000000, f000000f
# Experimento, falhas intermitentes
#nome, repetições, limite, bits, elf, inicio, fim, tipo, mascara, ativo, neutro
I1, 1, 10, bfi_test, intermitent, 00000800, 00001200, c0000000, 000033cc, 0000c350, 000124f8
I2, 1, 10, bfi_test, intermitent, 00000800, 000010f0, 40000000, ff0000ff, 00002528, 00001d4c
```

No modelo acima o caractere ‘#’ no início de uma linha indica um comentário e não será processado pelo controlador. A palavra “set” indica o nome do conjunto de campanhas. Nas linhas seguintes são mostradas quatro campanhas, duas envolvem falhas permanentes e duas envolvem falhas intermitentes.

Por exemplo, a primeira campanha, “P1” será executada apenas uma vez durante 10 segundos. A configuração do FPGA para este experimento está contida no arquivo “*bfi_test.bit*”. O Algoritmo de Injeção de Falhas está contido no arquivo “*permanent.elf*”. Os parâmetros seguintes indicam a faixa de endereços no qual a falha deve estar presente, entre 0x00001000 e 0x0001400. O penúltimo parâmetro indica o tipo de falha, 0x80000000 ao ser passado para o núcleo injetor configura uma falha onde as linhas indicadas do barramento devem ser forçadas para o valor ‘0’. Por fim, o último parâmetro é uma máscara binária que indica que as linhas 0x000ff000 do barramento devem ser forçadas para ‘0’.

A terceira campanha, “I1” também será executada uma vez, durante 10 segundos, utilizando a configuração “*bfi_test.bit*”. Os parâmetros seguintes indicam que o Algoritmo de Injeção de Falhas a ser utilizado está em “*intermitent.elf*”, a faixa de endereços válida vai de 0x00000800 a 0x000010f0, o tipo de falhas é 0xc0000000, uma falha onde as linhas 0xff0000ff tem seus valores invertidos logicamente. Os dois últimos parâmetros são os intervalos no tempo, em ciclos de *clock* nos quais a falha está presente ou não, 9512 ciclos e 7500, respectivamente.

4.3.2. Execução do Experimento

A execução do experimento pode ser controlada por um console ou automatizada por um *script Python* e segue os seguintes passos:

1. Criação de uma instância da classe controlador;
2. Configuração das interfaces de comunicação;
3. Inclusão de um ou mais conjuntos de campanhas;
4. Execução.

Um exemplo de um *script python* é apresentado abaixo:

```
#! /usr/bin/python

C = Controlador()
C.platform.core_port   = 'COM1'
C.platform.core_baud   = '57600'
C.platform.target_port = 'COM2'
C.platform.target_baud = '57600'
C.load('bfi_test_1')
C.load('bfi_test_2')
C.run()
```

4.3.3. Fluxo de execução

A execução dos experimentos pelo controlador segue o algoritmo apresentado abaixo para cada campanha de cada conjunto de campanhas:

1. Cria um diretório para os arquivos dos relatórios;
2. Configura o FPGA;
3. Carrega o algoritmo de injeção falha;
4. Executa um terminal serial para o sistema alvo;
5. Executa um terminal serial para o Módulo Injetor;
6. Transmite os parâmetros da campanha;
7. Contagem regressiva do tempo-limite; Recebe dados dos terminais;

8. Após o tempo-limite, interrompe os terminais;
9. Grava os arquivos de relatório.

4.3.4. Aquisição de Dados e Geração de Relatórios

A aquisição de dados do sistema alvo e do Módulo de Injeção é realizada através de portas seriais. A configuração das portas seriais é um atributo do experimento, uma vez que é esperado que em cada experimento sejam realizadas diversas campanhas com a mesma plataforma de hardware.

Os dados recebidos dos terminais seriais são gravados em arquivos texto em diretórios indexados pela hora de execução da campanha. O relatório do Módulo injetor possui informações sobre os parâmetros utilizados e os passos realizados em uma campanha de injeção de falhas. O relatório do sistema alvo é apenas uma cópia dos dados recebidos e que deverão ser analisados posteriormente com uma ferramenta adequada.

4.4. Projeto de um experimento

Esta seção descreve os passos necessários para a execução de um experimento de injeção de falhas completo.

1. **Desenvolvimento e validação do sistema alvo:** O sistema alvo deve ser estudado e implementado na plataforma de *hardware* escolhida. Este passo deve gerar os resultados ditos corretos para a análise posterior de falhas;
2. **Adaptação do módulo injetor e do sistema alvo:** Nesta fase, deve ser criado o projeto do Módulo Injetor, instanciando o Núcleo de Ccontrole e o Núcleo Injetor, apenas com o bloco de Controle de Alvo e o sistema alvo. O objetivo é construir a estrutura do experimento e verificar a temporização do sistema como um todo;

3. **Definição dos modelos de falhas:** Os modelos de falhas que serão utilizados no experimento devem ser definidos durante esta fase. Isso inclui criar os Algoritmos de Injeção Falhas. Cada configuração do FPGA nesta fase também deve ser validada com alguma aplicação sintética que verifique se as falhas injetadas são visíveis.
4. **Descrição das campanhas de injeção de falhas:** Neste ponto são descritas as campanhas de injeção de falhas que compõe o experimento.
5. **Execução do experimento:** O experimento é configurado e executado.
6. **Análise dos resultados:** Finalmente os relatórios gerados são recolhidos e analisados utilizando uma ferramenta especializada.

4.5. Conclusão

A plataforma de injeção de falhas apresentada aqui foi desenvolvida com base nos modelos de falhas e nas principais técnicas de injeção de falhas estudados nos capítulos anteriores. O conceito de comandos de compilador, utilizado em técnicas de injeção de falhas por simulação, foi transportado para a plataforma apresentada na forma dos Algoritmos de Injeção de Falhas utilizados pelo Módulo de Controle, enquanto o conceito de sabotadores foi utilizado no Mecanismo Injetor. A notação de modelos de falhas para barramentos levou a uma arquitetura modular, caracterizada pelos blocos funcionais do Módulo Injetor, onde o bloco Temporizador e o bloco Mecanismo injetor implementam, respectivamente, as características temporais e o efeito de determinado modelo de falha. Também é importante salientar o desenvolvimento do ambiente de software utilizado para a configuração e a automatização de experimentos de injeção de falhas.

5. Resultados Experimentais

Neste capítulo serão apresentados os experimentos utilizados para a validação do ambiente de injeção de falhas proposto. Primeiramente, cada bloco do Núcleo Injetor foi projetado e simulado separadamente. O segundo passo para a validação da plataforma proposta foi a execução de quatro experimentos de injeção de falhas em um processador *softcore*. Nos dois primeiros experimentos foram injetadas falhas em um barramento de memória. Os dois experimentos seguintes consistem em injetar falhas em um barramento associado a periféricos de entrada e saída. Em cada experimento serão utilizados diferentes modelos de falhas.

Após a etapa de validação experimental, foi realizada uma análise do *overhead* de área da plataforma de injeção de falhas proposta e um estudo comparativo entre a plataforma desenvolvida e as ferramentas propostas na literatura.

5.1. Verificação do núcleo injetor

Nesta seção serão apresentados os resultados da obtidos a partir da simulação de cada bloco. As simulações utilizam um sinal de *clock* de 100MHz.

5.1.1. Controle de Alvo

A Figura 5.1 mostra o resultado da simulação do bloco Controle de Alvo. Nessa simulação o bloco foi configurado para dividir o sinal de *clock* de entrada por 4. Os sinais de *pause* e *reset* foram acionados em um período arbitrário.

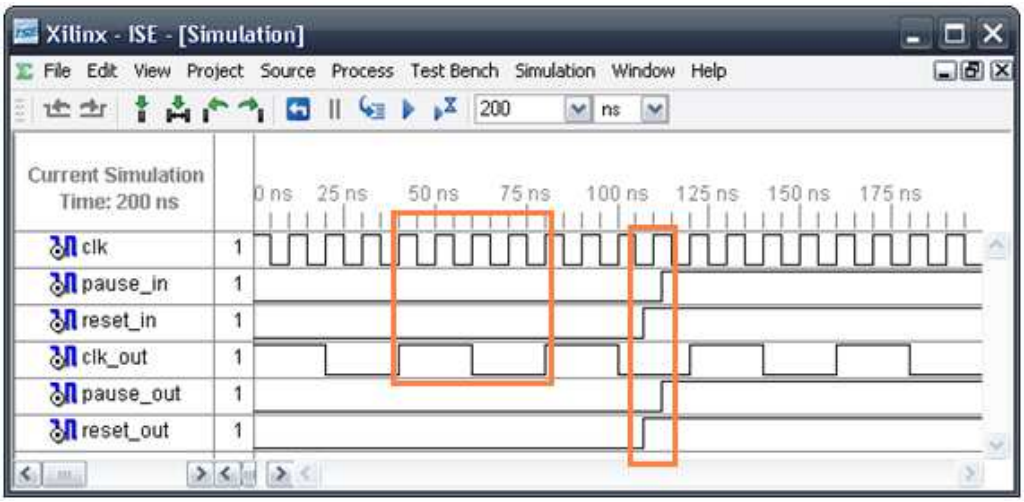

Figura 5.1 – Resultado da simulação para o bloco Controle de Alvo.

5.1.2. Comparador

A Figura 5.2 mostra o resultado da simulação para a implementação do bloco comparador utilizada nos experimentos seguintes. Este bloco recebe o barramento de endereço da memória do sistema alvo no sinal *address_in* e o sinal de controle de leitura/escrita no sinal *ctrl_in*. Este bloco foi configurado para capturar a faixa de endereços 0x00001000 – 0x00002000 utilizando os sinais *address_high* e *address_low*.

Quando o barramento apresentar um valor dentro da faixa de interesse e o sinal *ctrl_in* estiver em ‘1’, indicando uma operação de escrita, o sinal *c_ok* é colocado em ‘1’, sinalizando que o sistema alvo está em uma condição válida para a injeção de falhas.

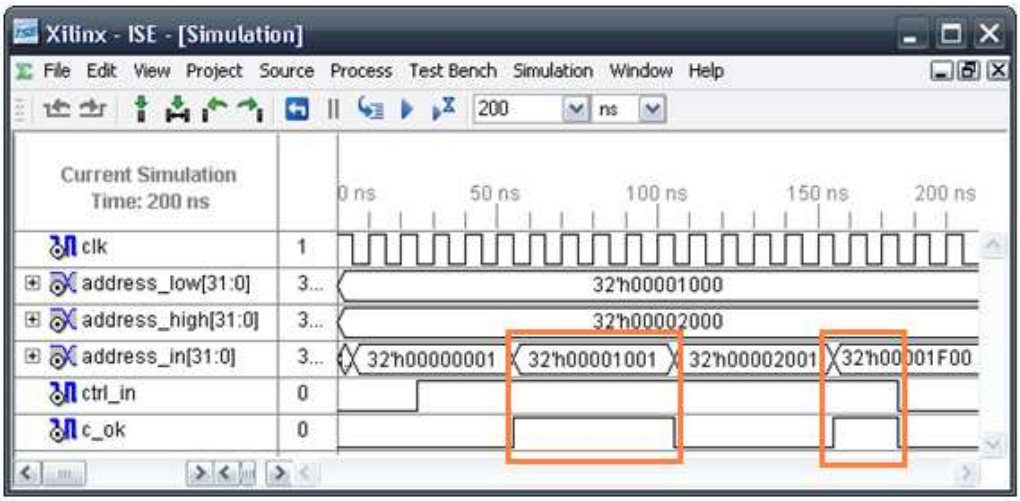

Figura 5.2 – Resultado da simulação para o Bloco Comparador.

5.1.3. Temporizador

A Figura 5.3 mostra o resultado da simulação do bloco Temporizador para o modo de operação permanente. Neste modo, o sinal *t_ok* terá o valor '1' sempre que o sinal *c_ok* também estiver em '1'.

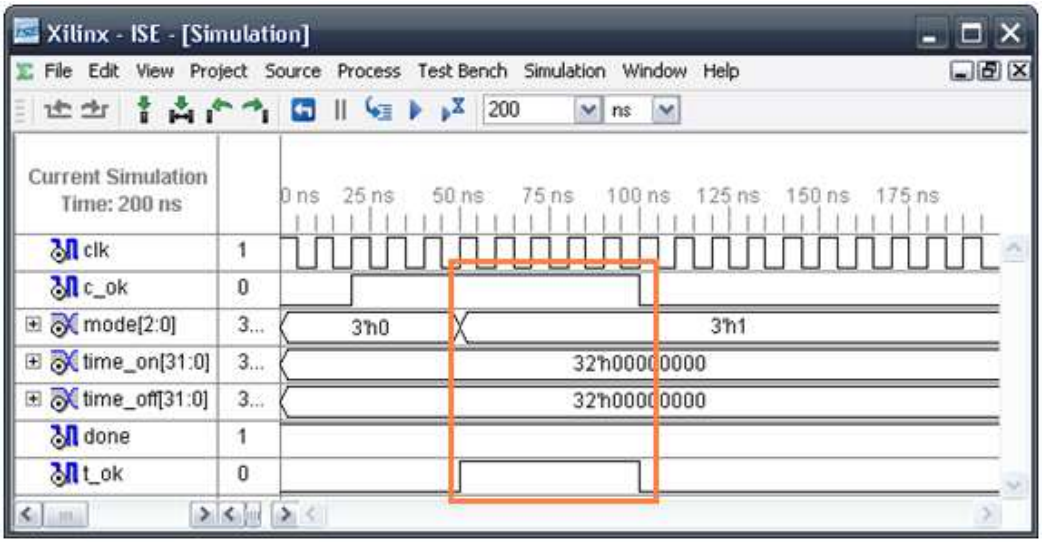

Figura 5.3 – Bloco Temporizador no modo de operação permanente.

A Figura 5.4 mostra o resultado da simulação do bloco Temporizador para o modo de operação falha transiente. Neste modo de operação, uma borda de subida no sinal *c_ok* dispara

um pulso positivo de um ciclo de *clock* de duração no sinal *t_ok*. A falha terá, portanto, duração de um ciclo de *clock*.

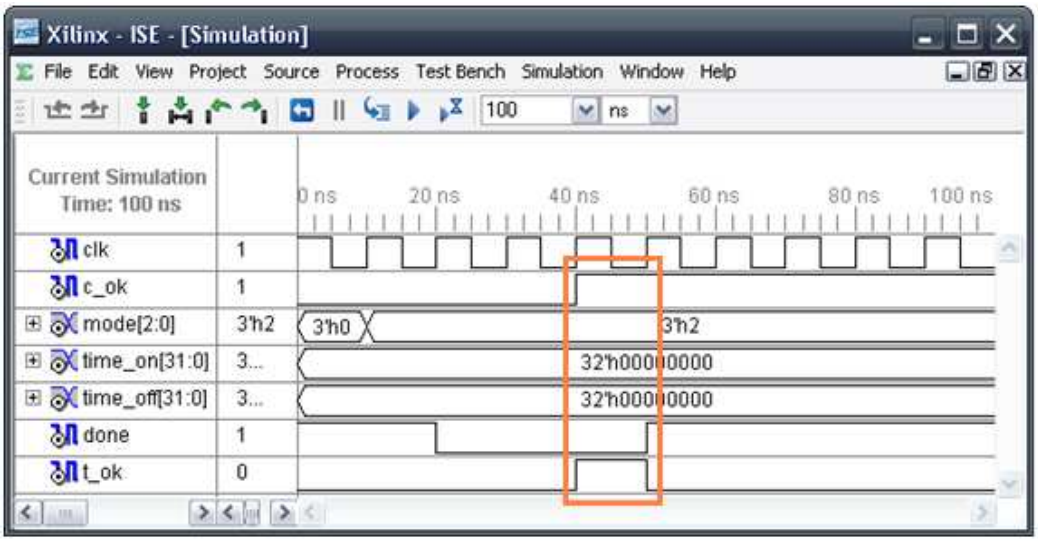

Figura 5.4 – Bloco Temporizador no modo de operação falha transiente.

A Figura 5.5 mostra o resultado da simulação do bloco temporizador para o modo de operação Falha Transiente Longa. Neste modo de operação, uma borda de subida no sinal *c_ok* dispara um pulso positivo de *time_on* ciclos de *clock* de duração no sinal *t_ok*.

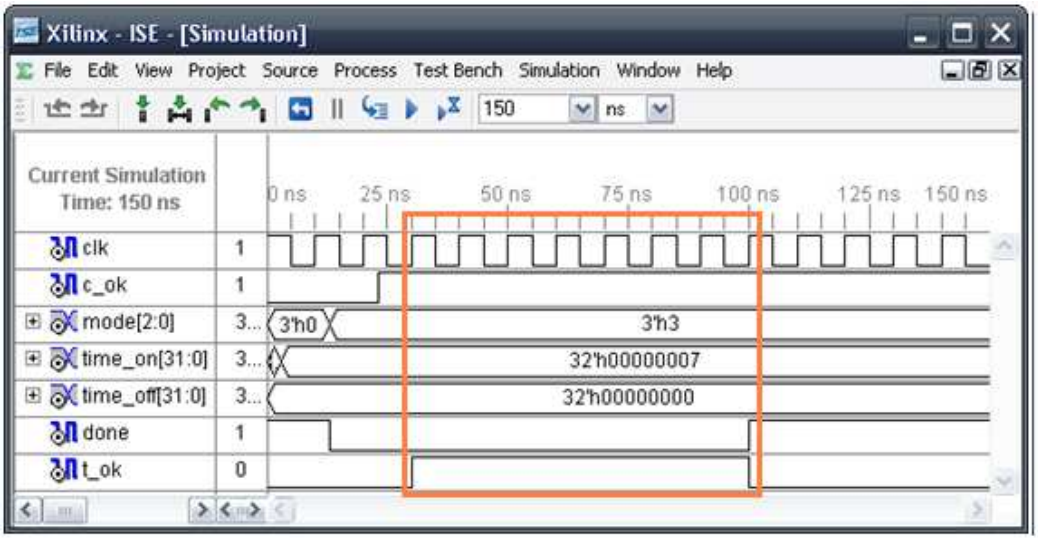

Figura 5.5 – Bloco Temporizador no modo de operação falha transiente longa.

A Figura 5.6 mostra o resultado da simulação do bloco Temporizador para o modo de operação cíclico. Neste modo de operação, uma borda de subida no sinal *c_ok* dispara um trem de pulsos no sinal *t_ok*. Este trem de pulsos possui *time_on* ciclos de *clock* com o valor ‘1’ *time_off* ciclos de *clock* com o valor ‘0’.

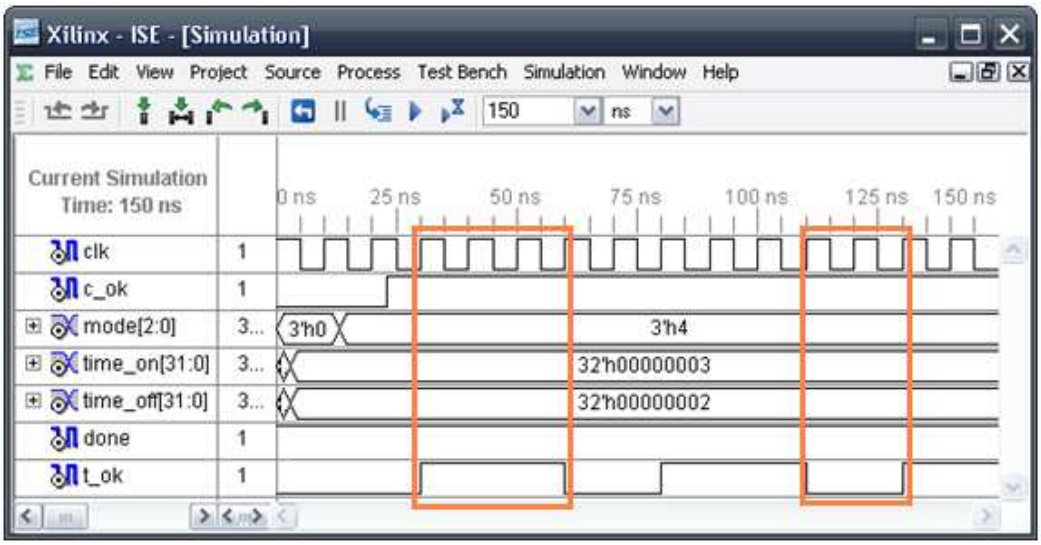

Figura 5.6 – Bloco Temporizador no modo de operação falha transiente longa.

5.1.4. Mecanismo Injetor

A Figura 5.7 mostra o resultado da simulação do bloco Mecanismo Injetor. Este bloco recebe o sinal *t_ok* do bloco Temporizador. O valor ‘1’ neste sinal habilita a injeção de falhas. O tipo de operação realizada no barramento *X* é dado pelo sinal *f_type*. Os bits a serem alterados são indicados pelos bits em ‘1’ correspondentes na máscara binária presente no sinal *mask*. Caso *f_type* receba “01”, os bits indicados serão forçados para ‘0’, caso receba “01”, estes bits serão forçados para ‘1’ e, finalmente, com “11” os bits terão seu valor invertido. O sinal *Y* recebe o resultado desta operação.

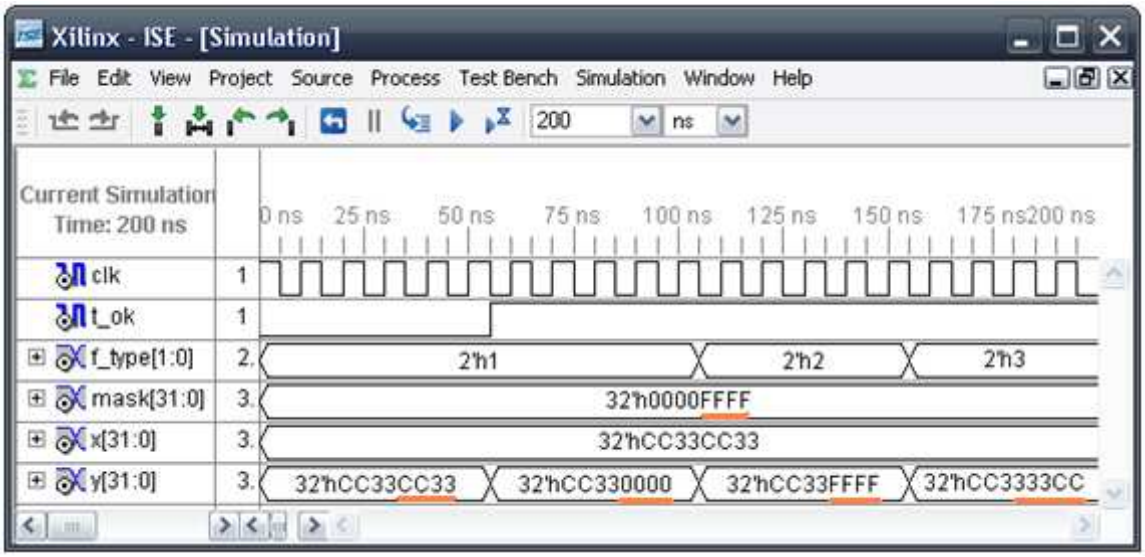

Figura 5.7 – Resultado da simulação para o bloco Mecanismo de Injeção.

5.2. Verificação da Plataforma

Nesta seção serão apresentados os experimentos realizados para verificar a plataforma de injeção de falhas como um todo.

5.2.1. Plataforma de hardware utilizada

A validação da plataforma desenvolvida e os experimentos deste trabalho foram realizados em uma placa de desenvolvimento NEXYS2, desenvolvida pela Digilent Inc. [27]. A Figura 5.8 mostra a vista superior da placa e a Figura 5.9 mostra sua arquitetura. Foram usados os pacotes de *software* Xilinx ISE e EDK 10.1i.

A placa NEXYS2 possui os seguintes componentes:

- FPGA Xilinx Spartan-3E, 1200k portas lógicas;
- 16 MB de memória PSDRAM;
- 16 MB de memória Flash;
- Memória de configuração do FPGA Xilinx Platform Flash ROM;
- Oscilador de 50MHz, mais um soquete para um segundo oscilador;

- 75 pinos de entrada/saída roteados para conectores de expansão, todos protegidos contra descargas eletrostáticas;
- Dispositivos de entrada/saída incluem 8 LEDS, 4 displays de sete segmentos, 4 botões e 8 chaves;
- Conectores incluem porta serial RS232, porta VGA e porta PS2;
- Fontes de alimentação chaveadas para todas as tensões (3V3, 2V5 e 1V2);
- Porta USB2 fornecendo alimentação, configuração dos dispositivos e transferência de dados em alta velocidade;
- Suporte aos pacotes de *software* ISE e EDK da Xilinx.

Figura 5.8 – Placa de desenvolvimento NEXYS2.

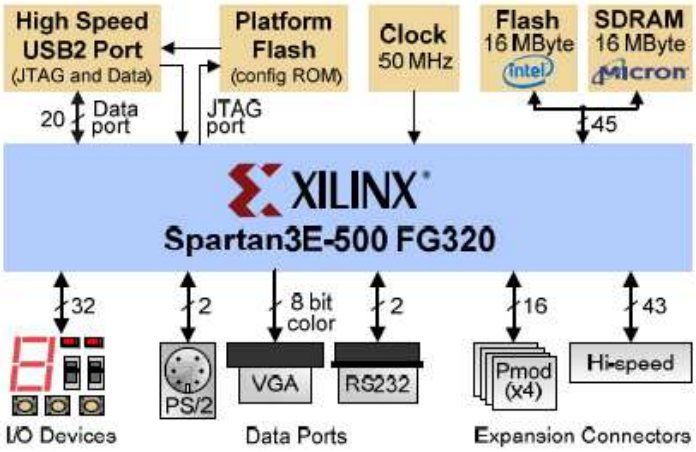

Figura 5.9 – Arquitetura interna da placa NEXYS2.

5.2.2. Sistema alvo: Processador Plasma

O *softcore* Plasma é um processador RISC de 32 bits, totalmente compatível com o conjunto de instruções da arquitetura MIPS I™, exceto pelas instruções de *load/store* desalinhadas [28][29].

Com três estágios de *pipeline* este processador, descrito em VHDL, possui diversos blocos que implementam funcionalidades como UART, *Timer*, além de controladores de interrupções, *Ethernet*, memória SRAM, memória DDR SDRAM e memória *Flash* [30].

A Figura 5.10 apresenta um diagrama de blocos da arquitetura básica deste processador embarcado.

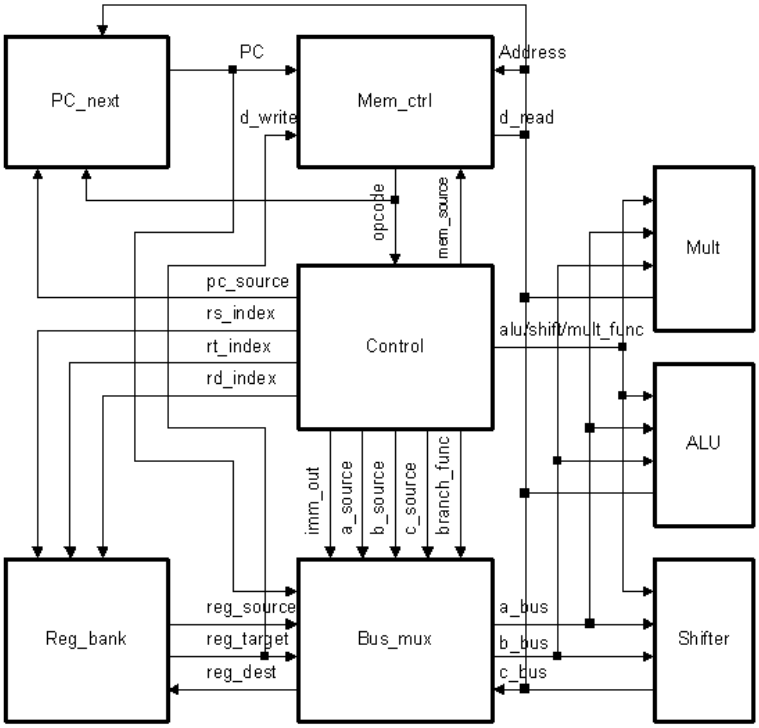

Figura 5.10 – Arquitetura do processador Plasma [30].

5.2.3. Carga de trabalho

Durante os experimentos, o sistema alvo deverá executar, como carga de trabalho, uma aplicação que torne evidente a ocorrência de uma falha. As aplicações utilizadas são descritas a seguir:

- *Walking ones*: A aplicação executada durante a injeção de falhas na memória do processador é um simples teste de memória do tipo walking ones. Este teste consiste em, sistematicamente, mover um bit com valor ‘1’ por todas as posições de uma palavra em cada endereço de memória. Em todos os outros bits é escrito o valor ‘0’. Primeiramente o processador percorre uma região de memória escrevendo este padrão de `1`s. Em seguida o processador volta para o primeiro endereço e inicia uma leitura desta região, esperando encontrar o mesmo padrão escrito anteriormente. Caso seja encontrado um valor diferente, é assumido que uma falha foi efetivamente injetada. Este procedimento é ilustrado no exemplo a seguir:

```
Write loop:
address     written
a:00001000  w:00000001
a:00001004  w:00000002
a:00001008  w:00000004
a:0000100c  w:00000008
a:00001010  w:00000010
a:00001014  w:00000020
a:00001018  w:00000040
a:0000101c  w:00000080
Read loop:
address     read        expected
a:00001000  r:00000001  e:00000001
a:00001004  r:00000002  e:00000002
a:00001008  r:00000004  e:00000004
a:0000100c  r:00000008  e:00000008
a:00001010  r:ffffffff  e:00000010 FAULT!!!
a:00001014  r:00000020  e:00000020
a:00001018  r:00000040  e:00000040
a:0000101c  r:00000080  e:00000080
```

Este programa percorre a região de memória de 0x00000800 até 0x00001800. Foi utilizada apenas uma porção vazia da memória, para que o próprio programa não fosse corrompido pelo experimento e acabasse por mascarar os resultados.

- Eco: Esta aplicação recebe caracteres ASCII do programa terminal, via comunicação serial e simplesmente os retorna pelo mesmo canal. Esses caracteres são enviados a cada 100ms. Caso o caractere recebido pelo terminal seja diferente do enviado, é assumido que uma falha foi efetivamente injetada. Foi necessária a modificação do programa terminal, para a comunicação com o sistema alvo. Os dados recebidos desta aplicação são ilustrados a seguir:

```
sent received
s:a  r:a
s:b  r:b
s:c  r:c
s:d  r:d
s:e  r:e
s:f  r:# FAULT!!!
s:g  r:g
s:h  r:h
```

5.2.4. Interface com o sistema alvo

A Figura 5.11 mostra como o sistema alvo foi ligado ao módulo injetor. A mesma configuração de FPGA foi utilizada para os quatro experimentos. Quatro instâncias do bloco núcleo injetor foram geradas, uma para cada barramento de interesse: barramento de leitura da UART, barramento de escrita da UART, barramento de leitura da RAM interna e, finalmente, o barramento de escrita da RAM interna. O barramento de endereços e sinais de controle associados foram ligados aos blocos comparadores, de forma que apenas uma faixa de endereços seja válida para injeção, nos experimentos que envolvem memórias. Nos experimentos relacionados ao periférico, somente o sinal de controle foi utilizado.

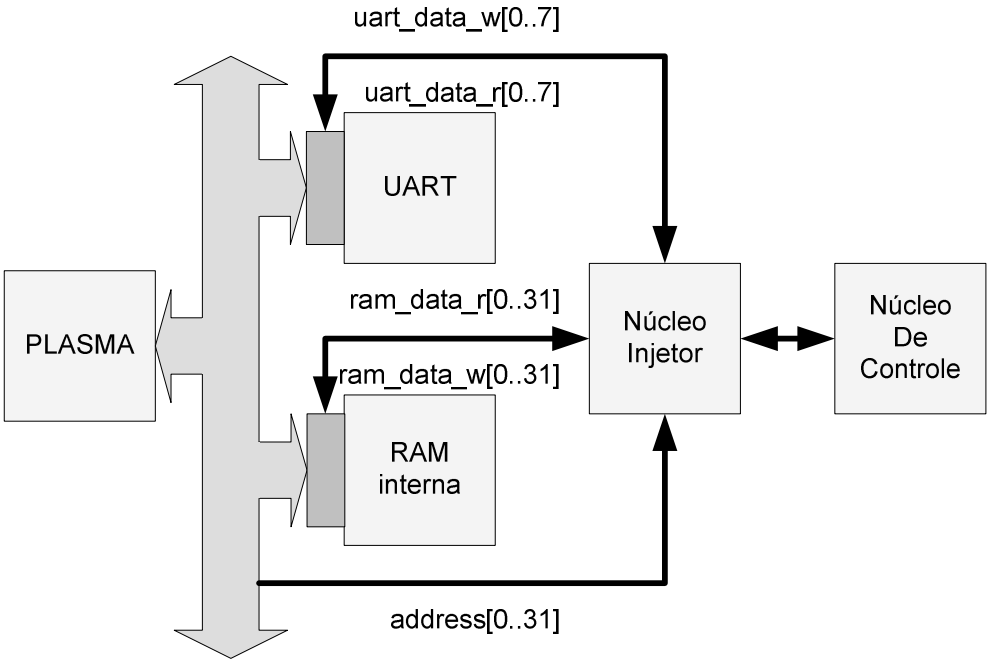

Figura 5.11 – Diagrama de blocos do processador Plasma com seus periféricos e os barramentos redirecionados para o núcleo injetor.

5.2.5. Experimento I – Barramento de Leitura da Memória

Este experimento utiliza os seguintes modelos de falhas:

- Efeito *Set0* com duração permanente, afetando o bit menos significativo do barramento, ativo na faixa de endereços que vai de 0x00001000 até 0x00001200.
- Efeito *Inv*, afetando os quatro bits menos significativos do barramento, ativo na faixa de endereços de faixa de endereços que vai de 0x00001200 até 0x00001400. Comportamento intermitente, com duração de 0x00001000 ciclos de *clock* e intervalos de 0x00002000 ciclos de *clock.*

Estes modelos de falhas são descritos pelo seguinte conjunto de campanhas:

```
set, Experimento I
Set0,     10, 10, exp, set_perm,    00001000, 00001200, 80000000, 00000001
Int flip, 10, 10, exp, intermitent, 00001200, 00001400, c0000000, 0000000f, 00001000, 00002000
```

Os resultados obtidos na a primeira campanha apresentam o seguinte padrão:

```
Write loop:
address     written
a:00001000  w:00000001
a:00001004  w:00000002
a:00001008  w:00000004
...
a:000017f8  w:40000000
a:000017fc  w:80000000
a:00001800  w:00000001
Read loop:
address     read        expected
a:00000800  r:00000001  e:00000001
a:00000804  r:00000002  e:00000002
a:00000808  r:00000004  e:00000004
a:0000080c  r:00000008  e:00000008
...
a:00001000  r:00000000  e:00000001 FAULT!!!
a:00001004  r:00000002  e:00000002
a:00001008  r:00000004  e:00000004
a:0000100c  r:00000008  e:00000008
a:00001010  r:00000010  e:00000010
...
a:0000087c  r:80000000  e:80000000
a:00000880  r:00000000  e:00000001 FAULT!!!
a:00000884  r:00000002  e:00000002
...
a:000011fc  r:80000000  e:80000000
a:00001200  r:00000000  e:00000001 FAULT!!!
a:00001204  r:00000002  e:00000002
...
a:0000127c  r:80000000  e:80000000
a:00001280  r:00000001  e:00000001 FAULT!!!
a:00001284  r:00000002  e:00000002
```

Este registro mostra que o bit menos significativo está sempre em zero e somente é percebido quando este bit está sendo verificado. Não são inseridas falhas fora da faixa de endereços especificada.

A segunda campanha do experimento gerou os seguintes resultados:

```
Read loop:
...
a:000011f0  r:10000000  e:10000000
a:000011f4  r:20000000  e:20000000
a:000011f8  r:40000000  e:40000000
a:000011fc  r:80000000  e:80000000
a:00001200  r:00000001  e:00000001
a:00001204  r:0000000d  e:00000002 FAULT!!!
a:00001208  r:00000004  e:00000004
```

```
a:0000120c  r:00000007  e:00000008 FAULT!!!
a:00001210  r:00000010  e:00000010
a:00001214  r:00000020  e:00000020
a:00001218  r:0000004f  e:00000040 FAULT!!!
a:0000121c  r:00000080  e:00000080
a:00001220  r:00000100  e:00000100
a:00001224  r:0000020f  e:00000200 FAULT!!!
a:00001228  r:00000400  e:00000400
a:0000122c  r:00000800  e:00000800
a:00001230  r:0000100f  e:00001000 FAULT!!!
a:00001234  r:00002000  e:00002000
a:00001238  r:00004000  e:00004000
a:0000123c  r:0000800f  e:00008000 FAULT!!!
...
a:000013f8  r:4000000f  e:40000000 FAULT!!!
a:000013fc  r:80000000  e:80000000
a:00001400  r:00000001  e:00000001
a:00001404  r:00000002  e:00000002
```

Observando os resultados obtidos, é possível concluir que a falha possui um caráter intermitente, com a falha aparecendo em intervalos regulares, e foi, portanto, injetada corretamente. É importante notar que o ciclo de falhas não precisa estar “em fase” com o ciclo de escrita, já que o período de injeção falhas não possui, necessariamente, a mesma duração do ciclo de escrita. Deve-se salientar que o processador Plasma apresenta uma peculiaridade: cada vez que um caractere é enviado para a porta serial, o processador é posto em estado de espera até que a transferência seja finalizada.

Nessas duas primeiras campanhas, as falhas foram injetadas no momento da escrita dos dados na memória, efetivamente inserindo uma falha na memória e corrompendo seu conteúdo. No caso do primeiro modelo de falhas utilizado, *Set0* permanente, podemos dizer que este comporta-se como um *stuck-at-0* na matriz de células da memória, pois o *bit* afetado sempre possuirá o valor ‘0’.

5.2.6. Experimento II – Barramento de Escrita da Memória

Este experimento utiliza os seguintes modelos de falhas:

- Efeito *Set1*, com duração permanente, afetando o bit mais significativo do barramento, ativo na faixa de endereços que vai de 0x00001400 até 0x00001600;

- Efeito *Set1*, afetando os dezesseis bits mais significativos do barramento, ativo na faixa de endereços que vai de 0x00001600 até 0x00001800. Será injetada apenas uma falha após 0x00002000 ciclos de *clock* com duração de 0x00001000 ciclos de *clock*.

Estes modelos de falhas são descritos pelo seguinte conjunto de campanhas:

```
set, Experimento II
Set1,       10, 10, exp, set_perm, 00001400, 00001600, 40000000, 10000000
Long-set-1, 10, 10, exp, long,     00001600, 00001800, c0000000, ffff0000, 00002000, 00001000
```

Os resultados obtidos na primeira campanha foram:

```
a:000013f8  r:40000000  e:40000000
a:000013fc  r:80000000  e:80000000
a:00001400  r:80000001  e:00000001 FAULT!!!
a:00001404  r:80000002  e:00000002 FAULT!!!
a:00001408  r:80000004  e:00000004 FAULT!!!
...
a:00001474  r:a0000000  e:20000000 FAULT!!!
a:00001478  r:c0000000  e:40000000 FAULT!!!
a:0000147c  r:80000000  e:80000000
a:00001480  r:80000001  e:00000001 FAULT!!!
a:00001484  r:80000002  e:00000002 FAULT!!!
...
a:000015f4  r:a0000000  e:20000000 FAULT!!!
a:000015f8  r:c0000000  e:40000000 FAULT!!!
a:000015fc  r:80000000  e:80000000
a:00001600  r:00000001  e:00000001 FAULT!!!
a:00001604  r:00000002  e:00000002 FAULT!!!
```

Nesta campanha, conforme foi configurado, o bit mais significativo está sempre em ‘1’. A falha não é percebida quando o padrão de bits correto é igual ao padrão gerado pela falha.

A segunda campanha gerou os seguintes resultados:

```
a:000015f8  r:40000000  e:40000000
a:000015fc  r:80000000  e:80000000
a:00001600  r:ffff0001  e:00000001 FAULT!!!
a:00001604  r:00000002  e:00000002
a:00001608  r:ffff0004  e:00000004 FAULT!!!
a:0000160c  r:00000008  e:00000008
a:00001610  r:00000010  e:00000010
a:00001614  r:ffff0020  e:00000020 FAULT!!!
a:00001618  r:00000040  e:00000040
```

```
a:0000161c  r:00000080  e:00000080
a:00001620  r:ffff0100  e:00000100 FAULT!!!
...
a:000017ec  r:ffff0000  e:08000000 FAULT!!!
a:000017f0  r:10000000  e:10000000
a:000017f4  r:20000000  e:20000000
a:000017f8  r:ffff0000  e:40000000 FAULT!!!
a:000017fc  r:80000000  e:80000000
a:00001800  r:00000001  e:00000001
Write loop >
```

É importante salientar que, diferentemente do experimento I, nestas duas últimas campanhas o conteúdo da memória é escrito corretamente e permanece, intacto, com o valor correto, já que as falhas são inseridas apenas no momento da leitura dos dados. Ainda assim, é possível dizer que o primeiro modelo de falhas utilizado, *Set1* permanente, também se comporta como um *stuck-at-1* na matriz de células da memória, pois independentemente do valor escrito, o *bit* afetado sempre é lido como se possuísse o valor '1'.

5.2.7. Experimento III – Barramento de Leitura do Periférico UART

Este experimento utiliza os seguintes modelos de falhas:

- Efeito *Inv*, com duração permanente, afetando o terceiro bit mais significativo;
- Efeito *Inv*, afetando todos os bits do barramento. Será injetada uma única falha transiente, após 0x00002000 cíclos de *clock*, com duração de um ciclo.

Estes modelos de falhas são descritos pelo seguinte conjunto de campanhas:

```
set, Experimento III
Permanent inv, 10, 10, exp, set_perm, c0000000, 00000020
Single inv,    10, 10, exp, single,   c0000000, 000000ff, 00002000
```

Os resultados obtidos na primeira campanha foram:

```
Sent received
s:a  r:A FAULT!!!
s:b  r:B FAULT!!!
s:c  r:C FAULT!!!
s:d  r:D FAULT!!!
s:e  r:E FAULT!!!
```

```
s:f  r:F FAULT!!!
s:g  r:G FAULT!!!
s:h  r:H FAULT!!!
s:i  r:I FAULT!!!
s:j  r:J FAULT!!!
s:k  r:K FAULT!!!
s:l  r:L FAULT!!!
s:m  r:M FAULT!!!
s:n  r:N FAULT!!!
s:o  r:O FAULT!!!
s:p  r:P FAULT!!!
s:q  r:Q FAULT!!!
s:r  r:R FAULT!!!
s:s  r:S FAULT!!!
s:t  r:T FAULT!!!
s:u  r:U FAULT!!!
s:v  r:V FAULT!!!
s:w  r:W FAULT!!!
s:x  r:X FAULT!!!
s:y  r:Y FAULT!!!
s:z  r:Z FAULT!!!
...
```

Uma falha permanente no barramento de leitura faz com que o padrão de bits correspondente a um caractere chegue corretamente no periférico e seja corrompido no momento de sua leitura pelo Plasma. Ao reenviar esse padrão corrompido, o terminal detecta a falha.

Os resultados obtidos na segunda campanha foram:

```
Sent received
s:a  r:a
s:b  r:b
s:c  r:c
s:d  r:d
s:e  r:š FAULT!!!
s:f  r:f
s:g  r:g
...
```

Neste experimento podemos ver apenas uma falha, conforme foi configurado ao utilizar, no bloco Temporizador, o modo de Falha Transiente. Este modo garante que a falha foi injetada no exato momento da leitura do caractere recebido.

Neste experimento, ambas as campanhas fazem com que o Plasma receba dados corrompidos do periférico. Qualquer tratamento posterior destes caracteres vai propagar este erro, possivelmente resultando num erro de dados.

5.2.8. Experimento IV – Barramento de Escrita do Periférico UART

Este experimento utiliza os seguintes modelos de falhas:

- Efeito *Set0* permanente, afetando o segundo bit mais significativo do barramento;
- Efeito *Inv*, afetando os quatro bits menos significativos do barramento. Comportamento intermitente, com duração de 0x00001000 ciclos de *clock* e intervalos de 0x00002000 ciclos de *clock*.

Descritos pelo seguinte conjunto de campanhas:

```
set, Experimento IV
Set0,     10, 10, exp, set_perm,    80000000, 00000040
Int_flip, 10, 10, exp, intermitent, c0000000, 0000000f, 00001000, 00002000
```

Os resultados obtidos na primeira campanha foram:

```
Sent received
s:a  r:! FAULT!!!
s:b  r:" FAULT!!!
s:c  r:# FAULT!!!
s:d  r:$ FAULT!!!
s:e  r:% FAULT!!!
s:f  r:& FAULT!!!
s:g  r:' FAULT!!!
s:h  r:( FAULT!!!
s:i  r:) FAULT!!!
s:j  r:* FAULT!!!
s:k  r:+ FAULT!!!
s:l  r:, FAULT!!!
s:m  r:- FAULT!!!
s:n  r:. FAULT!!!
s:o  r:/ FAULT!!!
s:p  r:0 FAULT!!!
s:q  r:1 FAULT!!!
s:r  r:2 FAULT!!!
s:s  r:3 FAULT!!!
s:t  r:4 FAULT!!!
s:u  r:5 FAULT!!!
s:v  r:6 FAULT!!!
s:w  r:7 FAULT!!!
s:x  r:8 FAULT!!!
s:y  r:9 FAULT!!!
s:z  r:: FAULT!!!
...
```

Na segunda campanha, os resultados obtidos foram:

```
Sent received
s:a  r:a
s:b  r:b
s:c  r:“ FAULT!!!
s:d  r:” FAULT!!!
s:e  r:• FAULT!!!
s:f  r:f
s:g  r:g
s:h  r:h
s:i  r:i
s:j  r:j
s:k  r:› FAULT!!!
s:l  r:œ FAULT!!!
s:m  r:? FAULT!!!
s:n  r:n
s:o  r:o
s:p  r:p
s:q  r:q
s:r  r:r
s:s  r:ƒ FAULT!!!
s:t  r:„ FAULT!!!
s:u  r:… FAULT!!!
s:v  r:† FAULT!!!
s:w  r:w
s:x  r:x
s:y  r:y
s:z  r:z
...
```

Analisando estes dois últimos experimentos, o padrão de falhas detectadas é muito similar ao padrão encontrado nos dois primeiros experimentos. De onde se conclui que a técnica de injeção de falhas proposta atua igualmente em barramentos de memória e barramentos de periféricos de um SoC.

5.3. Overhead de Área

A tabela 5.1 mostra a utilização dos recursos físicos do FPGA pelo processador Plasma e pelo sistema completo, ou seja, Plasma mais módulo injetor.

Podemos observar que o sistema completo consome aproximadamente o dobro de LUTs em relação ao sistema alvo, significando que ambos, sistema alvo e módulo injetor são, aproximadamente, do mesmo tamanho. O que era esperado, uma vez que ambos são processadores de 32 bits com capacidades semelhantes. O consumo relativamente maior de recursos de distribuição lógica pelo módulo injetor pode ser atribuído em parte pelos

barramentos internos do *Microblaze*, que são extremamente complexos, além das próprias interconexões necessárias à integração dos núcleos ao sistema alvo.

Tabela 5.1 – Recursos utilizados pelo processador Plasma.

Recurso	Presentes	Arquitetura			
		Plasma		Plasma + Módulo Injetor	
	[#]	[#]	[%]	[#]	[%]
Utilização lógica					
Slice Flip Flops	17,344	425	2%	2,988	17%
4 Input LUTs	17,344	3,426	19%	6,860	39%
Distribuição lógica					
occupied Slices	8,672	1,772	20%	4,413	50%
Slices containing only related logic	1,772	1,772	100%	4,413	100%
Slices containing unrelated logic	1,772	0	0%	0	0%
Total Number of 4 input LUTs	17,344	3,441	19%	6,980	40%
Number used as logic	-	3,170	-	6,217	-
Number used as a route-thru	-	15	-	120	-
Number used for Dual Port RAMs	-	256	-	512	-
Number of bonded IOBs	250	138	55%	10	4%
Number of RAMB16s	28	4	14%	8	28%
Number of BUFGMUXs	24	2	8%	3	12%

É importante salientar que o processador Plasma é um sistema relativamente pequeno e a proporção entre as áreas ocupadas pelo sistema alvo e o Módulo injetor tende a diminuir conforme o sistema alvo fica mais complexo. Obviamente o Módulo injetor só deve estar presente durante a fase de avaliação do sistema alvo, sendo retirado depois de validado o sistema.

5.4. Comparação entre atributos de outros sistemas

A tabela 5.2, mostra uma comparação entre a plataforma proposta e outras métodos de injeção de falhas em termos dos atributos descritos anteriormente.

Sobre a acessibilidade, a nova plataforma é baseada na alteração de constructos VHDL do tipo *signal*, utilizado para definir barramentos e registradores. Isso situa a plataforma proposta entre as técnicas de injeção de falhas em simulação, com uma acessibilidade total, e as técnicas de injeção de falhas em hardware.

O mecanismo de injeção de falhas utilizada na nova plataforma é baseado em conceitos provenientes das técnicas de injeção de falhas em simulação. O uso de algoritmos de injeção de falha pelo Núcleo de Controle pode ser vista como uma espécie de conjunto de comandos de simulador, resultando em uma controlabilidade similar a estas técnicas.

Tabela 5.2 – Comparação entre atributos de diferentes técnicas de injeção de falhas e plataforma proposta (adaptado de [15]).

Atributo	Simulação	Hardware			Software	
		com contato	sem contato	Plataforma Proposta	compilação	execução
Acessibilidade	Total	Pinos do integrado	Interno ao integrado	Barramentos	Registradores, Software	Registradores, I/O
Controlabilidade	Alta	Alta	Baixa	Alta	Alta	Alta
Custo	Baixo	Alto	Alto	Baixo	Baixo	Baixo
Intrusão	Nenhuma	Alta	Nenhuma	Baixa	Baixa	Alta
Portabilidade	Nenhuma	Baixa	Baixa	Alta	Baixa	Baixa
Repetibilidade	Alta	Alta	Baixa	Alta	Alta	Alta
Resolução temporal	Alta	Alta	Alta	Alta	Alta	Baixa
Risco	Baixo	Alto	Baixo	Baixo	Nenhum	Nenhum

Esta plataforma foi projetada, desenvolvida e validada utilizando uma placa de desenvolvimento comercial de baixo custo e um computador pessoal comum. Foram utilizadas ferramentas de software livre, como a linguagem *Python* e os compiladores GNU além da ferramenta Xilinx ISE, disponíveis gratuitamente. O custo maior ficou por conta da ferramenta Xilinx EDK, que permite o uso do processador *Microblaze*. Isto torna a nova plataforma mais cara que as técnicas de injeção de falhas por *software*, mas muito mais barata que as técnicas baseadas em *hardware*, que exigem a fabricação do mecanismo injetor e o uso de equipamentos especializados.

O nível de intrusão obtido pode ser considerado baixo, uma vez que o sistema alvo não tem seu comportamento ou suas características temporais modificadas com o uso da plataforma proposta. Este nível não pode ser mais baixo, pois é necessário o uso de uma plataforma de *hardware* que suporte a comunicação com o controlador do experimento e um FPGA que suporte o módulo injetor além do próprio sistema alvo, possivelmente limitando seu uso em protótipos

Conforme projetado, o uso de algoritmos de injeção de falhas parametrizáveis e a delegação das tarefas relativas à temporização para um *hardware* síncrono com o sistema alvo garantem uma alta repetibilidade e uma resolução temporal, comparáveis às abordagens de injeção de falhas em *hardware* com contato.

Por fim, o risco de danos ao equipamento é baixo, visto que os sistemas são apenas descrições em VHDL. Ainda assim, é possível que alguma combinação de fatores, o acionamento de algum periférico externo ao FPGA, por exemplo, seja comprometido pela ação da falha de forma que resulte em algum dano permanente à plataforma de *hardware*. Este risco, no entanto é inerente ao uso de técnicas de injeção de falha que envolvam o mundo físico.

6. Conclusão

Sistemas embarcados estão cada vez mais presentes nas mais diversas aplicações utilizadas nos mais variados segmentos de nossa sociedade. Estes sistemas estão sujeitos a falhas e é de nosso total interesse evitar que as mesmas tornem-se defeitos potencialmente desastrosos. Neste contexto, a avaliação de metodologias de teste e de técnicas de tolerância a falhas com o intuito de analisar a confiabilidade dos sistemas embarcados é extremamente importante. A avaliação através do uso de técnicas de injeção de falhas mostra-se um meio extremamente eficiente e confiável para atingir tais objetivos.

O estudo dos modelos de falhas permitiu a identificação dos principais comportamentos falhos no universo dos SoCs, bem como a individualização dos mecanismos de propagação dos mesmos no sistema alvo e como estes são percebidos pelo usuário. No que diz respeito aos modelos de falhas para barramentos, foi visto que os efeitos das falhas podem ser desacoplados de suas características temporais. Esta propriedade foi fundamental para o desenvolvimento modular do núcleo injetor proposto. Finalmente, Foi visto que a maioria das falhas em estruturas de uma memória pode ser mapeada para falhas presentes em sua matriz de células.

As diversas ferramentas de injeção de falhas estudadas mostram que é necessário promover um *trade-off* entre características desejáveis e efeitos indesejáveis no desenvolvimento das mesmas. Técnicas de injeção de falhas baseadas em simulação são capazes de observar e inserir falhas em praticamente qualquer ponto do sistema, de forma transparente, pois lidam apenas com modelos abstratos. Essa é justamente sua desvantagem, pois não é possível ter completa certeza de que o modelo representa o sistema final, depois de finalizado, operando em condições reais. Técnicas de injeção de falhas em *hardware*, de modo geral, possuem o comportamento oposto: são capazes de submeter o sistema a falhas no universo físico e, portanto, bastante realistas, mas perdem a fidedignidade dos resultados conforme aumenta seu alcance, pois implicam em maiores modificações no *hardware* para suportar mecanismos que

injetam falhas em pontos específicos do circuito. Por fim, técnicas de injeção de falhas via software não exigem modificações no *hardware*, mas alteram sutilmente seu comportamento de forma que podem inviabilizar experimentos em algumas situações, tais como aplicações em tempo real.

Tendo como base o estudo anterior, este trabalho teve como principal objetivo a especificação, implementação e validação de uma nova plataforma de injeção de falhas para a avaliação da confiabilidade agregadas a partir de metodologias de teste e técnicas de tolerância a falhas em *Systems-on-Chip*. Para cumprir estes objetivos foram necessários os seguintes passos:

1. Pesquisar técnicas de injeção de falhas presentes na literatura, buscando conceitos que poderiam auxiliar na obtenção da melhor combinação de características para o trabalho proposto e formas de implementar diferentes modelos de falhas;
2. Desenvolver um bloco sabotador capaz de alterar os sinais originais da forma mais transparente possível para o sistema alvo;
3. Desenvolver blocos auxiliares ao mecanismo de injeção de falhas que permitissem a correta aplicação da falha, no momento e no local correto, de acordo com a configuração dada pelo usuário, garantindo uma alta controlabilidade;
4. Desenvolver um núcleo de controle para operar o mecanismo de injeção de falhas conforme um procedimento especificado pelo usuário;
5. Desenvolver e integrar o ambiente de software necessário para o uso da ferramenta de injeção de falhas proposta, incluindo automatização dos experimentos e *softwares* auxiliares para comunicação com a plataforma de *hardware*;
6. Realizar campanhas de injeção de falhas para a validação da plataforma proposta utilizando os modelos de falhas apresentados anteriormente;
7. Comparação dos resultados experimentais com técnicas semelhantes propostas na literatura.

Assim, as principais conclusões obtidas neste trabalho foram as seguintes:

1. É possível realizar a injeção de falhas em SoCs complexos sem que seja necessário alterar de forma alguma o comportamento do sistema alvo. O uso dos sabotadores desenvolvidos garante a inserção de um atraso de propagação desprezível.
2. É evidente a relação existente entre complexidade e flexibilidade. A primeira versão da plataforma aqui proposta era bastante simples e compacta, mas somente era capaz de injetar falhas de acordo com um modelo previamente escolhido e adaptado ao sistema alvo. Conforme a necessidade de flexibilidade foi crescendo, a ferramenta foi tomando contornos cada vez maiores e mais complexos, chegando a ser comparável ao próprio sistema alvo.
3. O tamanho físico atingido pelo módulo injetor de falhas na implementação atual pode vir a ser um fator limitante para a sua adoção e uso, visto que as plataformas de desenvolvimento comerciais mais acessíveis costumam ter FPGAs relativamente pequenos. Da mesma forma, plataformas de desenvolvimento comerciais profissionais tendem a possuir FPGAs capazes de suportar pouco mais do que um sistema de tamanho médio para sua faixa de aplicação e preço.
4. Quando a técnica de injeção de falhas proposta é aplicada a barramentos de memórias, os modelos de falhas estão limitados àqueles definidos para memórias *word-oriented*, ou seja, falhas que afetam apenas uma palavra de dados de cada vez, pois o sabotador empregado somente tem acesso aos dados que estão presentes no barramento no momento do acesso a um determinado endereço, não sendo possível o acesso a dados armazenados em outros endereços.
5. Como conclusão final, a nova plataforma de injeção de falhas em SoCs mostrou-se eficaz na injeção de falhas de forma transparente para o sistema alvo, ou seja, não é necessário interromper o funcionamento do sistema alvo para injetar falhas. Por conseqüência, a plataforma desenvolvida pode ser utilizada para a injeção de falhas em um sistema de tempo real, sem comprometer as restrições temporais impostas pelo sistema. Assim, seu uso como alternativa às técnicas de injeção já existentes é viável.

6.1. Trabalhos futuros

Embora a implementação e validação da nova plataforma de injeção de falhas em SoCs tenha atingido muitos dos objetivos propostos, alguns pontos ainda necessitam de uma maior exploração, tanto pela complexidade envolvida quanto pela limitação do tempo. Como sugestão de trabalhos futuros, apresentamos alguns destes pontos:

- Implementação de uma ferramenta que automatize a inserção dos sabotadores no sistema alvo, visto que este é o ponto mais complexo da fase de adaptação da plataforma;
- Criação de um banco de dados para administração de configurações, campanhas e dados obtidos. A abordagem atual confia no usuário para manter o registro dos conteúdos dos diferentes arquivos correspondentes a configurações de FPGA, algoritmos de falhas e dados obtidos.
- É esperado que o universo de dados obtidos dos experimentos seja diverso demais para permitir a integração à plataforma de uma ferramenta genérica para análise dos resultados obtidos. No entanto, é sugerida a criação de uma metodologia que especifique cargas de trabalho padronizadas para o sistema alvo com a ferramenta correspondente para análise dos resultados.
- A criação de uma interface gráfica com o usuário capaz de facilitar ainda mais a criação e gerência de experimentos de injeção de falhas. Uma funcionalidade sugerida é a pré-seleção de um conjunto correto de combinações entre configurações de FPGA, algoritmos de injeção de falhas e os modelos de falhas pretendidos;
- Um estudo para a redução do *overhead* de área decorrente do uso de um processador similar aos sistemas alvos imaginados para a plataforma. Esta redução pode vir do uso de um processador com uma largura de dados menor, por exemplo 8 bits, ou do uso de um processador de 32 bits com funcionalidades reduzidas.
- Por fim, a plataforma proposta baseia-se na alteração de constructos VHDL do tipo *signal*. Este constructo é utilizado para a implementação de interconexões e registradores. Aqui foi explorado seu uso somente em interconexões, ficando

como sugestão a aplicação da técnica para injeção de falhas em registradores internos ao SoC.

7. Bibliografia

[1] BARDELL, P. H; MCANNEY, W. H; SAVIR, Jacob. *Built-in test for VLSI: pseudorandom techniques*, Wiley-Interscience, New York, NY, 1987.

[2] STROUD, E. C. A *Designer's Guide to Built-In Self-Test*. Boston : Kluwer Academic Publishers, pp. 15-27. 2002.

[3] LAPRIE J. C. *Dependable Computing and Fault Tolerance: Concepts and Terminology*. 15th IEEE Int. Sym. on Fault Tolerant Comp. Ann Arbor, Michigan, USA, pp: 2-11. 1985.

[4] GOOR, A. J. van der. *Using March Tests to Test SRAMs*. IEEE Design & Test of Computers 10(1): pp: 8-14. 1993.

[5] Nair et al. *Efficient Algorithms for Testing Semiconductor Random-Access Memories*. IEEE Trans. Computers 27(6): pp: 572-576. 1978.

[6] DEKKER, Rob; BEENKER, Frans P. M; THIJSSEN, Loek. *A realistic fault model and test algorithms for static random access memories*. IEEE Trans. on CAD of Integrated Circuits and Systems 9(6). pp: 567-572. 1990.

[7] ADAMS, R. Dean. *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test.* Kluwer Academic Publishers, Boston, 2003.

[8] BUSHNELL, M; AGRAWAL, V. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, Frontiers in Electronic Testing , Vol. 17. 1997.

[9] HAMDIOUI, S. *Testing Static Random Access Memories, Defects, Fault Models and Test Patterns,* Frontiers in Electronic Testing. Vol. 26. 1997.

[10] ZIMMER, Heiko; JANTSCH, Axel, *A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip,* Proceedings of the 1stIEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. Newport Beach, CA, USA. 2003.

[11] INTERNATIONAL Technology Roadmap for Semiconductors, 2001.

[12] LAJOLO et al. *Early Evaluation Of Bus Interconnects Dependability For System-On-Chip Designs*, Proceedings of the The 14th International Conference on VLSI Design (VLSID '01), p.371, January 03-07, 2001.

[13] CUVIELLO, Michael et al, *Fault modeling and simulation for crosstalk in system-on-chip interconnects*, Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design, p.297-303, San Jose, California, United States.1999.

[14] FAVALLI, Michele; METRA, Cecilia, *Bus crosstalk fault-detection capabilities of error-detecting codes for on-line testing*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.7 n.3, p.392-396, Sept. 1999.

[15] HSUE, Mei-Chen; TSAI, Timothy K; IYER, Ravishankar K. *Fault Injection Techniques and Tools*, Computer, v.30 n.4, pp.75-82, April 1997.

[16] KARLSSON, J. et al. *Integration and Comparison of Three Physical Fault Injection Techniques*. Pred. Depend. Comp. Sys. Vienna, Austria. pp: 309-329. 1995.

[17] KARLSSON, J. et al. *Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture*. 5th IFIP Work. Conf.on Depend. Comp. for Critic. App. Urbana- Champaign, USA. 1998;267-287.

[18] BENSO, A; PRINELLO, P. *Fault injection techniques and tools for embedded systems reliability evaluation*, Springer, 2003.

[19] JENN, E. et al. *Fault injection into VHDL models: the MEFISTO tool*, in: Proceedings of the 24th International Symposium on Fault-Tolerant Computing (FTCS-24), Austin, Texas (USA). pp. 66-75. 1994.

[20] KARLSSON, J. et al. *Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms*. IEEE Micro Magazine 14(1). Los Alamitos, CA, USA. pp: 8-23 1994.

[21] ARLAT, J; CROUZET, Y; LAPRIE, J. C. *Fault Injection for Dependability Validation of Fault-Tolerant Computer Systems*, Proc. 19th Ann. Int'l Symp. Fault- Tolerant Computing, IEEE CS Press, Los Alamitos, Calif., pp. 348-355. 1989.

[22] GUNNEFLO, O; KARLSSON, J; TONN, J. *Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation*, Proc. 19th Ann. Int'l Symp. Fault-Tolerant Computing, IEEE CS Press, Los Alamitos, Calif., pp. 340-347. 1989.

[23] SHOKROLAH-SHIRAZI, M; MIREMADI, S. G, *FPGA-Based Fault Injection into Synthesizable Verilog HDL Models*, Proceedings of the 2008 Second International Conference on Secure System Integration and Reliability Improvement, pp. 143-149. 2008.

[24] HAN, S; SHIN, K. G; ROSENBERG, H.A. *Doctor: An Integrated Software Fault-Injection Environment for Distributed Real-Time Systems*, Proc. Second Annual IEEE Int'l Computer Performance and Dependability Symp., IEEE CS Press, Los Alamitos, Calif. pp. 204-213. 1995.

[25] MICROBLAZE Processor Reference Guide Embedded Development Kit EDK 10.1i, Disponível em: <www.xilinx.com>, Acessado em 07/2009.

[26] SPARTAN-3 Generation FPGA User Guide Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families, Disponível em <www.xilinx.com>, Acessado em: 07/2009

[27] DIGILENT Nexys2 Board Reference Manual Disponível em: <www.digilentinc.com>, Acessado em 07/2009

[28] MIPS Technologies, Inc. MIPS Technologies. Disponível em: <http://www.mips.com/>, Acessado em 07/2009.

[29] ALECRIM, A. A. de; GARIBOTTI, R. F.. *Memória Cache em uma Plataforma Multiprocessada*. Faculdade de Engenharia / Faculdade de Informática. Porto Alegre : Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS, p. 67, Trabalho de Conclusão de Curso 2007.

[30] RHOADS, S. *Plasma CPU Core*. Disponível em <http://www.opencores.org/projects.cgi/web/mips/overview>, Acessado em: 07/2009.

[31] CARREIRA, J; MADEIRA, H; SILVA, J.G. *Xception: Software Fault Injection and onitoring in Processor Functional Units*, Proc. Fifth Ann. IEEE Int'l Working Conf. Dependable Computing for Critical Applications, IEEE CS Press, Los Alamitos, Calif., pp. 135-149. 1995.

[32] BIRNER, M; HANDL, T. *ARROW – A Generic Hardware Fault Injection Tool for NoCs*. Proc. Euromicro Conference Digital System Design, Patras, Grécia, PP. 465-472. 2009.

[33] GRUIAN, F. Testing for bridging faults. Disponível em <http://www.ida.liu.se/~flagr/work/doc/testpaper.pdf>, Acessado em 10/9/2009.

[34] NIRAJ IHA, S. G. Testing of Digital Systems, Press Syndicate of the University of

Cambridge, 2003.